Security Inference from Noisy Data

Li Zhuang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-32
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-32.html

April 8, 2008

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
08 APR 2008		**00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Security Inference from Noisy Data	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Electrical Engineering and Computer Sciences,Berkeley,CA,94720-1700**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**121**	

Security Inference from Noisy Data

by

Li Zhuang

B.E. (Tsinghua University) 2000
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor J. D. Tygar, Chair
Professor David Wagner
Professor Paul K. Wright

Spring 2008

The dissertation of Li Zhuang is approved:

Professor J. D. Tygar, Chair	Date
Professor David Wagner	Date
Professor Paul K. Wright	Date

University of California, Berkeley

Spring 2008

Security Inference from Noisy Data

by

Li Zhuang

Abstract

Security Inference from Noisy Data

by

Li Zhuang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor J. D. Tygar, Chair

My thesis is that contemporary information systems allow automatic extraction of security-related information from large amounts of noisy data. Extracting this information is the *security inference* problem: attackers or defenders extract information from noisy data that helps to compromise an adversary's security goals. I believe security inference is an important problem. Security inference often reveals a large amount of sensitive information that may be useful either to attackers or to system administrators. Attackers can use security inference to extract private information; system administrators can use security inference to determine the nature of attackers. Security inference is often a challenging problem because of the size and noisy nature of many real-world datasets. Our solution is to apply statistical analysis to this problem. We present two case studies that extract meaningful security knowledge from noisy data using statistical analysis. One goal is to explore selection of proper statistical analysis tools for security inference. The two case studies use a diverse set of statistical methods, which we believe to be applicable to other settings. We also propose a general framework for modeling security inference problems, which identifies key steps in the security inference process.

In the first case study, we examine the problem of *keyboard acoustic emanations*.

Attackers use security inference to analyze sound signals from typing on computer keyboards. We present a novel attack that takes as input a 10-minute sound recording of a user typing English text on a keyboard and recovers up to 96% of the characters typed. There is no need for a labeled training recording. Moreover, the recognizer bootstrapped this way can even recognize random text such as passwords: in our experiments, with 20 or fewer attempts to guess a random letter-only password, an attacker can guess 90% of 5-character passwords and 70% of 10-character password. This case study demonstrates that applying statistical analysis to security problems provides new tools for drawing powerful conclusions.

In the second case study, system administrators (or defenders) use security inference to determine the nature of attackers. We develop new techniques to map botnet membership and other characteristics of botnets using spam traces. The data consist of side channel traces from attackers: spam email messages received by Hotmail, one of the largest Web mail services. The basic assumption is that spam email messages with similar content often originate from the same controlling entity. These email messages share a common economic interest, so it is likely that a single entity also controls the machines sending these spam email messages. By grouping spam email messages with similar content and determining the senders of these email messages, one can infer the composition of the botnet. This approach can analyze botnets regardless of their internal organization and means of communication. This work also reports new statistics about botnets.

In this thesis, we leverage recent developments in the areas of applied data mining, statistical learning, and distributed data analysis. The approaches we discuss are easily deployable to real systems.

Professor J. D. Tygar, Chair Date

Acknowledgements

I would like to thank my advisor J. D. Tygar for introducing me to the area of security and privacy and guiding me through my research. I benefited from the encouragements I received from him to pursue interesting research ideas, or simply to to keep improving writing or presentation. His trust and support allowed me to explore research problems that intrigue me most. This work would not have been materialized without his valuable guidance.

I would like to thank Professor David Wagner for much good advice and help during my Ph.D. study. I would like to thank Professor Paul K. Wright for serving on my qualifying exam and dissertation committee. His early feedback to my thesis proposal was very helpful. I would like to thank Ben Y. Zhao for help on Cashmere and other projects, and help with how to conduct research in general. I would like to thank Marco Barreno for useful feedback on drafts of this thesis and previous projects. I would like to thank Feng Zhou for insightful discussions during the keyboard emanations study and the Cashmere project. I would like to thank John Dunagan, Dan Simon, Helen Wang, Ivan Osipkov and Geoff Hulten from Microsoft Research for collaborating and supporting in the botnet detection project.

I would like to thank the following people for enlightening discussions and help while I was at Berkeley: Wei Xu, Jimmy Su, Yitao Duan, Ling Huang, Qi Zhu, Chris Karlof, Yaping Li, Hao Chen, Jingtao Wang, Hao Zhang, Rachna Dhamija and Monica Chew.

Portion of the work described in this thesis were first reported in conference papers [Zhuang *et al.*, 2005; Zhuang *et al.*, 2008].

This work was supported in part by the National Science Foundation through CITRIS (NSF award number EIA-01225989) and the TRUST (Team for Research

in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), Cisco, British Telecom, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies. The opinions expressed here are my own and do not necessarily reflect the views of the US government, the National Science Foundation or any other funding sponsors.

Dedicated to my husband, Feng

Contents

List of Figures

List of Tables

Chapter 1

Introduction

My thesis is that contemporary information systems allow automatic extraction of security-related information from large amounts of noisy data. Extracting this information is the *security inference* problem: attackers or defenders extract information from noisy data that helps to compromise an adversary's security goals. Security inference is often a challenging problem because of the size and noisy nature of many real-world datasets. The bulk of this dissertation comprises two case studies in security inference using statistical and data mining approaches. The application of statistical analysis methods is a relatively recent development in security. One goal of this work is to explore the selection of proper statistical analysis tools for security applications. We also propose a general framework for modeling security inference problems using the two cases as examples. In the first case study, we present a method to recover typed characters from the sound of typing on keyboards. In this case, we show that by analyzing the sounds from typing on computer keyboards, attackers can compromise user security and privacy. In the second case study, we present a novel approach to identifying botnets and monitoring their behavior by analyzing spam email. When botnets run spam campaigns to generate profit, they leave traces of

their identities in spam email as *side effects*. The second case shows that defenders use side effects in attacker activity to extract information that helps to improve security. These two case studies illustrate that security inference can be used either by attackers or by defenders. Attackers can use it to extract private information; defenders can use it to determine the nature of attackers.

In the first case study, we examine the problem of *keyboard acoustic emanations*. Attackers use security inference to analyze sound signals from typing on computer keyboards. We present a novel attack that takes as input a 10-minute sound recording of a user typing English text on a keyboard and recovers up to 96% of the characters typed. There is no need for a labeled training recording. Moreover, the recognizer bootstrapped this way can even recognize random text such as passwords: in our experiments, with 20 or fewer attempts to guess a random letter-only password, an attacker can guess 90% of 5-character passwords and 70% of 10-character passwords.

Our work removes the requirement of labeled training samples, implying keyboard emanation attacks are more serious than previous work suggests. We test our approach in both quiet and noisy environments, recovering significant information about the typing in both settings. We successfully apply our approach to three different models of keyboards, suggesting that many keyboards are vulnerable to this attack. We find that the length of recording necessary to recover typed characters can be as short as five minutes.

The key insight in this case study is that typed text is usually not random: most typing is text in English or another language, and has a highly structured form (in particular, a relatively small number of common words and word combinations). Our attack uses the statistical properties of the English language to reconstruct text from sound recordings without any labeled training data. The attack uses a combination of standard machine learning and speech recognition techniques, including cepstrum features, hidden Markov models, linear classification, and feedback-based incremen-

tal learning. Practical technical challenges in this work include modeling language constraints, addressing errors in clustering algorithms, and automatically improving recovery rate over time. Chapter 3 details how our work addresses these challenges to mount a successful attack.

In the second case study, system administrators (or defenders) use security inference to determine the nature of attackers. We develop new techniques to map botnet membership and other characteristics of botnets using spam traces. The data consist of side channel traces from attackers hidden in spam email messages received by Hotmail, one of the largest Web mail services. A single large botnet can participate in many *spam campaigns* (coordinated mass mailings of spam). The basic assumption is that spam email messages with similar content often originate from the same controlling entity. These email messages share a common economic interest, so it is likely that a single entity also controls the machines sending them. Each email message records in its *header* the entire relay chain from the sender to the destination, which can reveal the sender of each message. By grouping spam email messages with similar content and determining the senders of these email messages, we can infer the composition of the botnet.

As we show in Chapter 4, this case study contrasts with most previous studies of botnets. The statistical perspective yields several benefits. First, the email data is readily available for analysis. Second, the approach is easier to deploy than previous approaches. Third, by focusing analysis on features tied to the economic motivations behind many botnets, it is harder for botnet controllers to evade detection than with previous approaches.

We detect 294 botnets and about 460 thousand bots in the spam email traces and identify a set of important metrics for measuring botnets and their activities. We find that more than half of botnets that send spam contain over 1000 machines. Botnets use only some of their machines in each campaign: more than 80% of botnets use

fewer than half of their machines in each campaign. Large botnets send fewer spam messages per bot, which makes them harder to detect using volume-based rules. This work also shows that some botnets are dedicated to the spamming business: 60% of botnet-related spam is from long-lived botnets. We also find that botnets are widely distributed over the Internet: more than half of the botnets we study contain machines from at least 30 countries. Statistical analysis provides us with insight into the characteristics of botnets.

1.1 Side Channel Information

The relentless improvement of computing technology and expansion of networking infrastructure have changed many aspects of society. One particular change is the creation and retention of tremendous amounts of digital data, along with the means to access the data from anywhere, anytime. Important and sensitive information is being digitized, stored and sent over the Internet, intentionally or unintentionally. For example: people routinely communicate using VoIP or video conferencing, sending audio and video data of themselves and their surroundings; Internet services, such as search engines and email services, collect data reflecting people's intentions, needs, and other personal traits; server systems generate log data, which contain information about the systems. The data are often *noisy*, containing uncertain, contradictory, or incorrect information, alongside useful and correct information.

The explosion in the amount of data has security and privacy implications. It presents both challenges and opportunities for the security community. On the one hand, digitization and increased availability of data could result in unintentional leaks of personal or secret information. Seemingly innocuous data can contain hidden sensitive information. Attackers now have more data available, so they will extract more information from it. On the other hand, more data can mean better monitoring,

analysis, and tools for catching the attackers. The large quantity and diversity of data could make it easier to identify the source of problems accurately at an early stage.

The information we use in this dissertation is *side channel information* or *side effects*. In cryptography, *side channel information* refers to information gained from the physical implementation of a cryptosystem rather than theoretical weakness in the algorithm. A related concept is *side effects*: a function or expression produces a side effect if it modifies some state (such as static variables or files on disk) in addition to returning a value. When talking about a computer system (for example, a desktop computer or a large distributed system such as the Internet), we generalize the concepts of both *side channel information* and *side effects* to mean any information that leaks out where such exposure is not explicitly indicated in the system design. As we show in Chapter 2, side channels such as timing information, resource consumption, unprotected emanations, and unintended usage of certain data can provide the necessary information for security analysis.

1.2 Challenges in Side Channel Information Analysis

Several trends in hardware, software and services indicate the need for new methods for security analysis of side channel information.

- *More types of information are available for analysis.* In recent years, new consumer electronic products have entered the market, such as the Apple iPod and the Microsoft Zune. These new devices and others, such as smart cards and RFID chips, introduce new ways of communication and sharing data. For example, the Nike+iPod Sport Kit is a new wireless accessory for the iPod Nano. The Kit transmits information from a sensor on the user's shoes to a receiver on the iPod when the user runs or walks. The receiver and iPod interpret

that information and provide interactive audio to the user about his workout. This type of information exposure is new and requires new forms of security analysis [Saponas *et al.*, 2007]. Furthermore, new types of Internet services are appearing and traditional services such as email are increasing in use. People depend heavily on email, search engines, e-commerce, online banking, instant messaging, social websites, and other Internet services in everyday life. These new services have enjoyed wide adoption before security researchers have had the opportunity to thoroughly study the main channels of information exchanged and exposed by such systems, not to mention the possible of side channels.

- *More interconnections arise between different types of information.* The situation is complicated as data collected by different services are connected. For example, a user may use the same username and password at different websites, such as an online banking website and a web forum. Because different websites provide different levels of security protection to users, a weakly protected forum website could endanger the user's banking account. Unfortunately, without careful study of these connections, these security threats remain hidden. In another example (see Section 3.6.1), when a person types on a keyboard, the combination of acoustic information and timing information may reveal typed characters. (As we discuss later, it is an open problem to determine whether the combination of timing and acoustic information yields more information than timing or acoustic information alone.)

- *Data collected from side channels are noisy in nature.* As byproducts from a running system, data collected from side channels are noisy. For example, sound recordings have some level of environmental noise. Each email message in any given spam campaign can be slightly different because of random text. Noisy data is harder to analyze. Data collected from side channels can be inconsistent,

can depend on invisible internal states, or can be insufficient to describe parts of the state space. Data analysis requires developing methods to extract the most important information and eliminate noise. While it is infeasible to eliminate noise completely from data, statistical analysis algorithms are sufficiently robust to tolerant some noise. For example, Fast Fourier Transform (FFT) features and cepstrum features capture important frequency information from acoustic signals emitted by keyboards even with significant noise in the signal (see Section 3.4.1).

- *The ability to process massive datasets has become a necessity.* As large-scale distributed systems and Internet services become widely available, datasets collected from these systems (including side channel information) can exceed the processing ability of a single machine. Infrastructure and tools for processing massive datasets are key requirements for performing security analysis in many cases. For example, Hotmail receives thousands of terabytes of spam email each day. Even a down-sampled subset requires hundreds of gigabytes per day to preserve meaningful results. Without proper infrastructure and tools, just one day's dataset requires weeks of computation on a single machine and infeasible storage capacity. I present more details about data processing in Section 4.4.2.

These trends demonstrate the need for new approaches to security problems that are data-centric, especially when data are collected from side channels.

1.3 Towards Statistical Analysis of Side Channel Information

In this thesis, we leverage methods in statistical learning and recent developments in data analysis infrastructures to address the challenges above.

We focus on features that are hidden in the data and find techniques to extract them reliably. In most data collected from side channels, different occurrences of the same event often have small variations each time. Statistical analysis can overcome these small variations and find the inherent consistency across occurrences. For example, we show how to use a statistical learning algorithm called Expectation Maximization (EM) to infer character keystrokes from the varying sounds they emit (see Section 3.4).

When the data come from multiple domains, we focus on the features that connect instances from all domains. One way to approach inter-domain connections is similar to the "join" operation in database queries. The features common across two datasets serve as *foreign keys* in database to join feature vectors from both domains. This operation expands the feature vector of each sample. We apply statistical analysis to these expanded feature vectors to find inherent connections, and group the samples from the same event. Another way to approach the inter-domain connections is to aggregate information from each domain. We first apply statistical learning methods to features in each domain separately, extracting inherent structure from each domain. We then combine aggregated knowledge about the same event at a higher level. This mechanism appears in the second case study, where we combine aggregated knowledge from instant messenger log and spam email traces to overcome the problem of a single computer having different IP addresses at different times. (see Section 4.4.5).

To process large datasets that exceed the capacity of a single machine, we leverage cluster infrastructure for our data analysis. Specifically, we rewrite statistical learning algorithms using existing APIs for parallel processing. For example, to process the giant dataset of spam email messages, we use the Dryad [Isard *et al.*, 2007] distributed infrastructure APIs as discussed in Section 4.4.2.

In practice, we also find that domain specific knowledge is important to represent features in a format that facilitates further processing.

1.4 Contributions of this Work

In this thesis, we study the problem of security inference, or inferring high-level security-related knowledge automatically from large, noisy datasets. Specifically, we propose a data mining approach to security inference problems and use two case studies to demonstrate the feasibility of this approach. This work makes the following contributions.

- We demonstrate a method that combines data mining, statistical analysis, and new infrastructure tools to create a practical, scalable, general approach for security analysis of large noisy datasets. We demonstrate how to select appropriate statistical tools for different types of side channel information, then we show that these techniques are practical in today's hardware and software environment.

- This work generalizes the study of side channel information. Most research in the past has used side channel information in attacks only. Side channel information can be especially useful to attackers because side channel data collection easily escapes the notice of defenders. However, the same properties that make side channel information useful to an attacker can also make it useful to the defender. In this thesis, we study side channel information in a broader context. When defenders collect side channel information leaked from attacking systems, they can use it to defend against attacks and improve security.

- The case study of keyboard acoustic emanations presents important results. We are able to recover up to 96% of characters from the sound of typing on computer keyboards. This implies that keyboard acoustic emanation attacks are far more serious than previous work [Asonov and Agrawal, 2004] suggests. We build keystroke classifiers directly from sound recordings without any pre-

labeled sound samples, using only the assumption that the typing represents English text. A keystroke classifier bootstrapped this way can recognize any random sequence of characters, such as passwords. This case study demonstrates that applying statistical analysis to security problems provides new tools for drawing powerful conclusions.

- The case study of spam email provides a new way to study the botnet problem. We study botnets from the perspective of the economic motivations of botnet controllers. This work is the first to analyze the behavior of entire botnets (in contrast to individual bots) from spam email messages. Because of the economic perspective we use in this study, our approach can analyze botnets regardless of their internal organization and means of communication, and it is not thwarted by encrypted traffic or customized botnet protocols, unlike previous work using IRC trackers [Cooke *et al.*, 2005; Freiling *et al.*, 2005] or DNS lookup [Rajab *et al.*, 2006; Ramachandran *et al.*, 2006; Rajab *et al.*, 2007]. This work also reports new classes of information about botnets. For example, we report on the relationship between botnet usage and basic properties such as size. We also confirm previous reports on capabilities of botnet controllers and botnet usage patterns.

In summary, this thesis studies the security inference problem — in particular, side channel information analysis — in a practical way. We leverage recent developments in areas of applied data mining, statistical learning, and distributed data analysis infrastructure. The approaches we discuss in this thesis are easily deployable to real systems.

Chapter 2

Side Channel Information and Its Security Implications

In this chapter we present an informal categorization of side channel information and its security implications. This provides context for the later concrete discussion.

2.1 Timing Information

Timing information refers to the duration of a event or the time it takes between events in a sequence of events.

Previous research has used timing information to attack cryptosystems. Timing attacks exploit the fact that different cryptographic operations, and even the same operation with different inputs, can take different amounts of time to execute. By measuring the amount of time the system takes to respond various queries, it is possible to infer operations and parameters in a cryptosystem. Generally, timing attacks are used to target a specific implementation of a cryptosystem.

For example, Brice Canvel et al. show how to attack OpenSSL versions prior to 0.9.6c [Canvel *et al.*, 1993]. They attack the implementations by exploiting the

way OpenSSL pads messages. Each message contains the raw message, a message authentication code (MAC), and padding. OpenSSL encrypts each message as a whole in transmission and adds padding after the MAC, since messages must have a specified length. The padding in this specific implementation has a specific form: if 2 bytes are added then the padding is "1,1", a three-byte padding is "2,2,2", etc.. To decrypt the message, OpenSSL first checks the padding and then computes the MAC only after determining that the padding is correct. If the padding is not correct, the protocol exits without computing the MAC. The vulnerability is that the amount of time consumed is significantly different depending on whether the padding is correct or incorrect, because checking the MAC is a relatively time consuming operation. Using this fact, Brice Canvel et al. implement a timing attack for the case of cipher block chaining (CBC) mode:

> The attack assumes that multiple SSL or TLS connections involve a common fixed plaintext block, such as a password. An active attacker can substitute specifically made-up ciphertext blocks for blocks sent by legitimate SSL/TLS parties and measure the time until a response arrives: SSL/TLS includes data authentication to ensure that such modified ciphertext blocks will be rejected by the peer (and the connection aborted), but the attacker may be able to use timing observations to distinguish between two different error cases, namely block cipher padding errors and MAC verification errors. This is sufficient for an adaptive attack that finally can obtain the complete plaintext block. [Canvel *et al.*, 1993].

Kocher describes a timing attack exploiting the execution time for square-and-multiply algorithm used in modular exponentiation, which depends linearly on the number of "1" bits in the key [Kocher, 1996]. Private key operations in RSA [Rivest *et al.*, 1983], Diffie-Hellman [Diffie and Hellman, 1976] and many other cryptographic

algorithms involve computing $R = y^x \bmod n$, where n is public, x is the private key and y can be selected by an attacker. To mount an attack, the attacker asks the victim to compute $y^x \bmod n$ for several careful chosen values of y and measures the response time for each operation. With enough queries, the attacker can recover information about secret key x.

More recently, Boneh and Brumley have demonstrated a network-based remote timing attack using Chinese Remainder Theorem optimization on SSL-enabled web servers [Brumley and Boneh, 2003]. They exploit two algorithmic data dependencies in OpenSSL that cause timing variations in RSA decryption. They test their attack in local network environment with low network latency and successfully recover a server's private key in a matter of hours.

Two different timing attacks by Percival [Percival, 2005] and Bernstein [Bernstein, 2005] exploit extra time caused by cache misses. In Hyper-Threading processors, threads share not only processor resources such as execution units but also access to the memory caches. Shared caches can create a cryptographic side channel, for example, through virtual memory paging. Loading a page from disk on a cache miss takes longer than the time to return from a cache hit. This can leak a bit of information from one process to another. L1 and L2 cache misses also create similar information channels. Shared access to memory caches therefore permits a malicious thread to monitor the execution of another thread, enabling in many cases the theft of cryptographic keys.

2.2 Consumption of Certain Resources

Computer systems, especially real-time systems, consume a certain amount of resources during execution. The amount of resources consumed can provide information about the internal states of a system. The resources consumed could be power,

network bandwidth, storage, memory, etc. It is often possible to monitor consumption of resources from outside of a system and use these observations to infer states inside a system.

As one concrete example, consider the power consumption of a cryptographic hardware device (such as a smart card or microchip). If the amount of power consumed by different cryptographic operations varies, statistical analysis may be able to correlate the differences with inputs or stored cryptographic keys, causing power consumption to leak crucial information about the cryptosystem.

Kocher et al. demonstrate that an attacker can compute intermediate values of data blocks and key blocks by statistically analyzing data collected from multiple cryptographic operations [Kocher *et al.*, 1999]. They propose two power monitoring attacks. The first one, called Simple Power Analysis (SPA), directly interprets power consumption variations due to sequences of cryptographic operations. When continuously monitoring power consumption across a single cryptographic operation, the 16 rounds of the Data Encryption Standard (DES) [Wikipedia, 2008a] are clearly distinguishable. SPA uses the power consumed in each round to infer the sequence of instructions executed. This analysis can reveal information in cryptographic implementations in which the execution path depends on the data being processed, such as the DES key schedule, DES permutations, comparisons, multipliers, and exponentiators.

The second, more powerful attack is Differential Power Analysis (DPA). The power consumption is correlated with the data values being manipulated, through these variations are smaller and can be overshadowed by measurement errors and noise. DPA uses statistical functions tailored to the target algorithm to analyze power consumption. Signal processing and error correction techniques can extract secrets from measurements that are too noisy to be analyzed by Simple Power Analysis (SPA).

2.3 Unprotected Emanation Channels

In the context of computer security, emanations traditionally refer to electric or electromagnetic radiation produced by electronic equipment. Emanations from electronic devices have long been a topic of concern in the security and privacy communities [Briol, 1991]. Emission Security or Emanations Security (EMSEC) has been a national security concern as well. For example, TEMPEST is a U.S. government code word that identifies a classified set of standards for limiting electric or electromagnetic radiation emanations from electronic equipment.

In public research, Kuhn has recovered the display on CRT and LCD monitors using indirectly reflected optical emanations [Kuhn, 2002; Kuhn, 2003]. In 2002, Kuhn showed that the intensity of the light emitted by a raster-scan screen as a function of time corresponds to the video signal convolved with the impulse response of the phosphors. Enough high-frequency content remains in emitted light to permit reconstruction of readable text by deconvolving the signal received with a fast photosensor. The optimal emanations from cathode-ray tube (CRT) displays are recoverable even after distortion or diffuse reflection from a wall at distances from tens up to hundreds meters.

In 2003, Kuhn further showed that modern flat-panel displays can be at least as vulnerable to electromagnetic eavesdropping [Kuhn, 2003], especially when connected by the Digital Video Interface (DVI) cables: even shielded cables leak detectable radio waves to the environment. The serial transmission of DVI cables effectively modulates the video signal in ways that provide an eavesdropper with better reception quality than VGA cables. Kuhn discusses several new attacks and defenses involving emanations from flat-panel displays and DVI cables.

Acoustic emanations provide another source of information. Researchers have shown that acoustic emanations of matrix printers carry substantial information

about the printed text [Briol, 1991]. Some researchers suggest it may be possible to discover CPU operations using acoustic emanations [Shamir and Tromer, 2004].

Another example of acoustic emanations is the sound emitted while typing on computer keyboards, which is a side effect of using a computer. This is the subject of Chapter 3 of this dissertation. Here is a brief overview — Two observations provide clues about recovering information from these emanations. First, different keystrokes can sound different given the non-uniformity of the keyboard supporting plate and individual keys. Second, people tend not to type random text: for example, English speakers type English most of the time. By exploiting these two observations, attackers who overhear the acoustic emanations of typing can recover the actual characters typed.

2.4 Unintended Usage of Information

Some systems log certain information during execution. When combined with other sources, information logs can be used in ways unintended by their designers.

Frankowski et al. demonstrate a method for associating user identities across their blogs and forum discussions [Frankowski *et al.*, 2006]. People sometimes publish reviews of movies on blogs or forums to share opinions with others. However, this information might also be used for the unintended and perhaps undesirable purpose of linking identities. A person may write opinionated reviews about movies in a blog under a pseudonym while participating in a forum or web site for scholarly discussion without a pseudonym or anonymity. It is often possible to link these separate identities because different blogs or forums show common features such as mention of certain movies, authors, and journal articles.

Attackers can also leak information when they use systems to mount attacks. Some attackers compromise machines remotely and control illicit networks (called

botnets) of these machines. Botnet controllers typically want to hide information about propagation and command-control traffic, to avoid discovery and neutralization of compromised machines. In Chapter 4, we discuss our hypothesis that the economic motivation behind such networks almost always leaves traces to be found. For example, when botnets send spam email, similarities between spam email messages can reveal information about botnet identities and botnet membership of the source computer. The connections among spam email messages form a type of side channel beyond the design specifications of botnets. We show in Chapter 4 that security can be enhanced by analyzing these hidden connections. In this case, the identities of email senders are logged as part of the SMTP protocol. This log information is used to infer botnet membership, which is a different purpose from its intended use.

Chapter 3

Side Channel System Attack: Keyboard Acoustic Emanations

3.1 The Case: Keyboard Acoustic Emanations

Emanations produced by electronic devices have long been a topic of concern in the security and privacy communities [Briol, 1991]. Sound of typing on computer keyboards is a type of acoustic emanations, called keyboard acoustic emanations. Keyboard acoustic emanations leak information about keystrokes typed. This chapter reports on recovering keystrokes typed on a keyboard from a sound recording of the user typing.

Acoustic keyboard emanations, are not uniform across different instances, even when the same device model is used, and they are affected by the environment. Different users on a single keyboard or different keyboards (even of the same model) emit different sounds, making reliable recognition hard [Asonov and Agrawal, 2004]. Asonov and Agrawal achieved relatively high recognition rate (approximately 80%) when they trained neural networks with text-labeled sound samples of the same user

typing on the same keyboard. Their attack is analogous to a known-plaintext attack on a cipher – the cryptanalyst has a sample of plaintext (the keys typed) and the corresponding ciphertext (the recording of acoustic emanations). This *labeled training* sample requirement suggests a limited attack, because the attacker needs to obtain training samples of significant length. Presumably these could be obtained from video surveillance or network sniffing. However, video surveillance in most cases should render the acoustic attack irrelevant, because even if passwords are masked on the screen, a video shot of the keyboard could directly reveal the keys being typed.

In this chapter we argue that a labeled training sample requirement is unnecessary for an attacker. This implies keyboard emanation attacks are more serious than previous work suggests. The key insight in our work is that the typed text is often not random. When one types English text, the finite number of mostly used English words limits possible temporal combinations of keys, and English grammar limits word combinations. One can first cluster (using unsupervised methods) keystrokes into a number of acoustic classes based on their sound. Given sufficient (unlabeled) training samples, a *most-likely mapping* between these acoustic classes and actual typed characters can be established using the language constraints.

This task is not trivial. Challenges include: 1) How can one mathematically model language constraints and mechanically apply them? 2) In the first sound-based clustering step, how can one address the problem of different keys clustered in the same acoustic class and a single key clustered in multiple acoustic classes? 3) Can we improve the accuracy of the guesses by the algorithm to match the level achieved with labeled samples?

Our work answers these challenges, using a combination of machine learning and speech recognition techniques. We show how to build a keystroke recognizer that has better recognition rate than labeled sample recognizers in [Asonov and Agrawal, 2004]. We only use a sound recording of a user typing.

Our method can be viewed as a machine learning version of classic attacks to simple substitution ciphers. Assuming the ideal case in which a key produces exactly the same sound each time it is pressed, each keystroke could be easily given an acoustic class according to the sound. The acoustic class assignment would be a permutation of the key labels. This is exactly an instance of substitution cipher. Early cryptographers developed methods for cryptoanalyzing substitution ciphers. Our attack can be viewed as an extension of these methods – but our problem is more difficult because the sound of a particular keystroke varies even when it is produced by the same typist.

We built a prototype that can bootstrap the recognizer from about 10 minutes of English text typing, using about 30 minutes of computation on a desktop computer with a Pentium IV 3.0G CPU and 1GB of memory. After the bootstrap step, it could recognize language-independent keystrokes in real time, including random keystrokes occurring in passwords, with an accuracy rate of about 90%. When language-dependent constraints are applied to English text, we achieve a 90-96% accuracy rate for characters and a 75-90% accuracy rate for words.

The purpose of this chapter is to show that after applying machine learning techniques to side channel information, the types of information leak regarded as safe before can be sources of severe attacks. We also show in this chapter about how to select proper combinations of feature extraction and learning algorithms in order to perform statistical analysis.

3.2 Related Work

We briefly review two related previous research studies examining recovery of keystrokes, each using a different type of side channel information.

To the best of our knowledge, Asonov and Agrawal were the first researchers

to publish a concrete attack exploiting keyboard acoustic emanations [Asonov and Agrawal, 2004]. They note that the sound of keystrokes differ slightly from key to key. They give a concrete method to recover information about typing on keyboards, using neural networks as acoustic classifiers. Their approach is to first "teach" the neural networks about what the different keys sound like. To do this, each key is typed 100 times. The neural network is trained with the label (the key being typed) and the corresponding sound. The raw digitalized sound input is too large for their neural networks, so each keystroke is represented as a vector of Fast Fourier Transform (FFT) features. The trained neural network then can be used to recognize subsequent keystrokes.

Based on the *supervised learning* approach above, Asonov and Agrawal show:

- A wide variety (e.g. different keyboards of the same model, different models, different brands) of keyboards have keys with distinct acoustic properties.
- Sound recordings from as far away as 15 meters suffice for neural network supervised learning if sophisticated microphones such as parabolic microphones are used.
- Their neural network supervised learning is sensitive to training errors: if input label are inaccurate, their recognition rates drop sharply. The effectiveness of the approach also depends a lot on the comprehensiveness of the training samples, i.e. whether it contains enough samples for each key or not.

Asonov and Agrawal's work opened a new field. However, there are limitations in their approach:

- Their attack is for *labeled* acoustic recordings. Their attack works well only with the same settings (i.e. the same keyboard, person, recording environment, etc.) as the training recording, and such training data are hard to obtain in

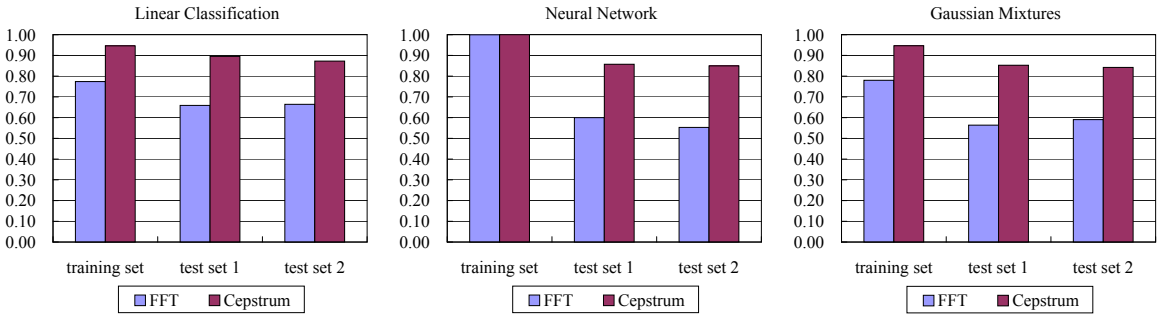

Figure 3.1: Recognition rates using FFT and cepstrum features. The Y axis shows the recognition rate. Three different classification methods are used on the same sets of FFT or cepstrum features.

many cases. Training on one keyboard and recognizing on another keyboard of the same model yields much lower accuracy rates, at around 25%. Even if we count all occasions when the correct key is among the top four candidates, the accuracy rate is still only about 50%. Lower recognition rates are also observed when the system is trained for one typist and then applied to another typist.

- The set of acoustic classification techniques used leaves room for improvement. In our work, we found features superior to FFT and acoustic classifiers superior to neural networks. Figure 3.1 compares FFT and cepstrum features and also compares three classifiers: linear classification, neural networks and Gaussian mixtures. The classifier is trained on the *training set* data and is then used to classify the training set itself and two other data sets. Character recognition rate using cepstrum features (discussed below) on average is better than character recognition using FFT. This is true for all data sets and classification methods. Neural networks perform worse than linear classification on the two test sets. In this experiment, we could only approximate the experiment settings in [Asonov and Agrawal, 2004]. But the significant performance differences indicate that there are better alternatives to FFT and neural networks combination.

Timing information is a different type of side channel information related to keyboard typing. Timing information includes the time between two keystrokes, the time between keystroke push to keystroke release, etc. Song, Wagner and Tian showed how to extract information based on the time between two consecutive keystrokes [Song *et al.*, 2001]. They considered interactive login shells encrypted with the SSH protocol. In this scenario, an eavesdropper can detect the time between consecutive keys. Statistical analysis shows that the distribution of time between a pair of keys vary for different key pairs. Contributing factors include: whether keys are typed with alternating hands or the same hand, with different fingers or the same fingers, etc. The types of pairs defined in their work capture the physical distances between keys and also the the response time of human beings. However, many different pairs may belong to the same type, e.g. two letters typed by alternating hands. Timing information is generally not helpful in distinguishing different pairs in the same type. Their work gives some analysis of the amount of information leaked by timing information. In Section 3.6.1, we give an approach to combine timing information with our acoustic emanation recognition. However, to date we have only observed modest improvements by adding timing information. It remains an open question whether the two methods together can yield substantially higher recognition rates.

3.3 The Attack

In this section, we present an overview of our attack. Section 3.4 presents the attack in full.

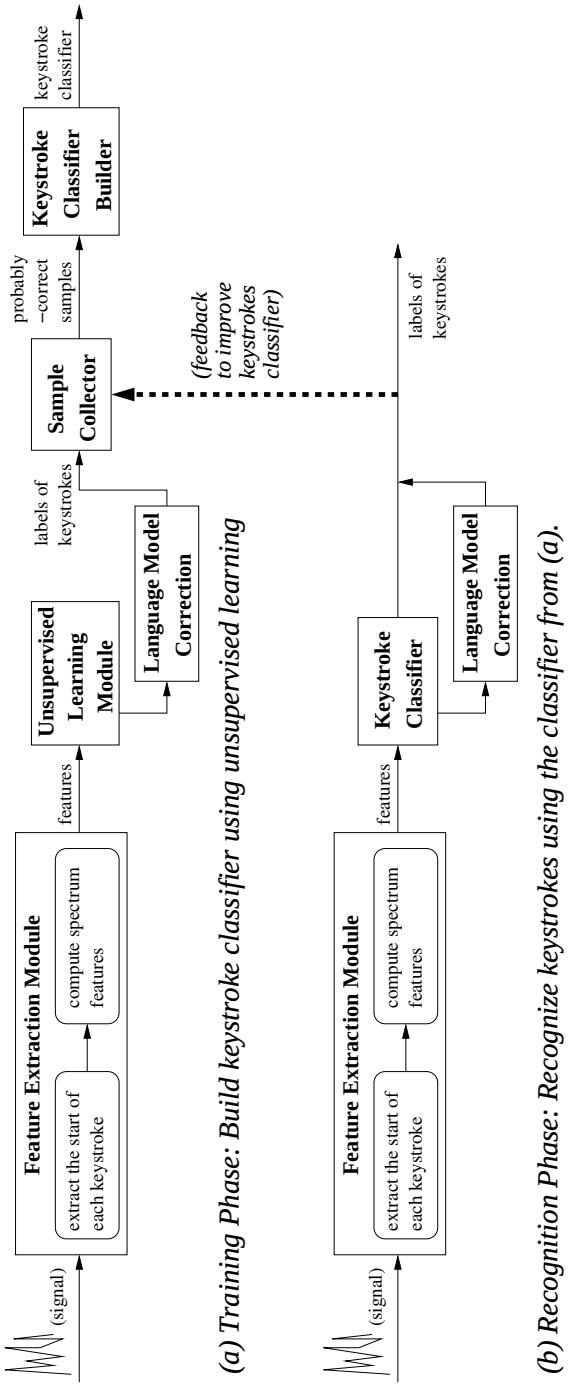

Figure 3.2: Overview of the attack

We take a recording of a user typing English text on a keyboard, and produce a recognizer that can, with high accuracy, determine subsequent keystrokes from sound recordings if it is typed by the same person, with the same keyboard, under the same recording conditions. These conditions can easily be satisfied by, for example, placing a wireless microphone in the user's work area or by using parabolic or laser microphones from a distance. Although we do not necessarily know in advance whether a user is typing English text, in practice we can record continuously, try to apply the attack, and see if meaningful text is recovered.

Figure 3.2 presents a high level overview of the attack.

The first phase (Figure 3.2(a)) trains the recognizer. It contains the following steps:

- *Feature extraction.* We use cepstrum features, a technique developed by researchers in voice recognition [Childers *et al.*, 1977]. As we discuss below, cepstrum features give better results than FFT.

- *Unsupervised key recognition* using unlabeled training data. We cluster each keystroke into one of K acoustic classes, using standard data clustering methods. K is chosen to be slightly larger than the number of keys on the keyboard. As we discuss in Section 3.1, if these acoustic clustering classes correspond exactly to different keys in a one-to-one mapping, we can easily determine the mapping between keys and acoustic classes. However, clustering algorithms are imprecise. Keystrokes of the same key are sometimes placed in different acoustic classes and conversely keystrokes of different keys can be in the same acoustic class. We let the acoustic class be a *random variable* conditioned on the actual key typed. A particular key will be in each acoustic class with a certain probability. In well clustered data, probabilities of one or a few acoustic classes will dominate for each key. Once the conditional distributions of the acoustic classes are

determined, we try to find the most likely sequence of keys given a sequence of acoustic classes for each keystroke. Naively, one might think picking the letter with highest probability for each keystroke yields the best estimation and we can declare our job done. But we can do better. We use a Hidden Markov Model (HMM) [Rabiner and Juang, 1986]. HMMs model a stochastic process with state. They capture the correlation between keys typed in sequence. For example, if the current key can be either "h" or "j" (e.g. because they are physically close on the keyboard) and we know the previous key is "t", then the current key is more likely to be "h" because "th" is more common than "tj". Using these correlations in English[1], both the keys and the key-to-class mapping distributions can be efficiently estimated using standard HMM algorithms. This step yields accuracy rates of slightly over 60% for characters, which in turn yields accuracy rates of over 20% for words.

- *Spelling and grammar checking.* We use dictionary-based spelling correction and a simple statistical model of English grammar. These two approaches, spelling and grammar, are combined in a single Hidden Markov Model. This increases the character accuracy rate to over 70%, yielding a word accuracy rate of about 50% or more. At this point, the text is quite readable (see Section 3.4.3).

- *Feedback-based training.* Feedback-based training produces a keystroke acoustic classifier that does not require an English spelling and grammar model, enabling random text recognition, including password recognition. In this step, we use the previously obtained corrected results as labeled training samples. Note that our corrected results are not 100% correct. We use heuristics to select words that are more likely to be correct. For examples, a word that is *not* spell-

[1]Other languages than English have different probabilistic distributions of pairs, but the method still applies.

corrected or one that changes only slightly during correction in the last step is more likely to be correct than those that had more changes. In our experiments, we pick out those words with fewer than 1/4 of characters corrected and use them as labeled samples to train an acoustic classifier. The recognition phase (Figure 3.2(b), described below) recognizes the training samples again. This second recognition typically yields a higher keystroke accuracy rate. We use the number of corrections made in the spelling and grammar correction step as a quality indicator. Fewer corrections indicate better results. The same feedback procedure is performed repeatedly until no significant improvement is seen. In our experiments, we perform three feedback cycles. Our experiments indicate both linear classification and Gaussian mixtures perform well as classification algorithms [Jordan, 2008], and both are better than neural networks as used in [Asonov and Agrawal, 2004]. In our experiments, character accuracy rates (without a final spelling and grammar correction step) reach up to 92%.

The second phase, the recognition phase, uses the trained keystroke acoustic classifier to recognize new sound recordings. If the text consists of random strings, such as passwords, the result is output directly. For English text, the above spelling and grammar language model is used to further correct the result. To distinguish between two types of input, random text or English text, we apply the correction and see if the result is close to English text.

In practice, a human attacker can typically determine if text is random. An attacker can also identify occasions when the user types user names and passwords. For example, password entry typically follows a URL for a password protected website. Meaningful text recovered from the recognition phase *during an attack* can also be fedback to the first phase. These new samples along with existing samples can be used together to increase the accuracy of the keystroke classifier. Our recognition

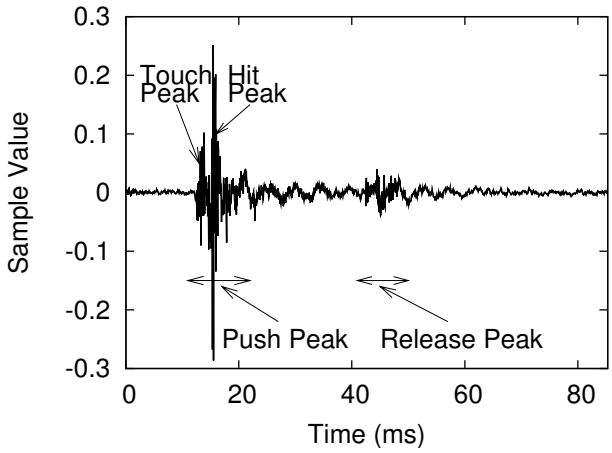

Figure 3.3: The audio signal of a keystroke

rate improves over time (see below sections).

3.4 Technical Details

Below, we describe in detail the steps of our attack. Some steps (feature extraction and supervised classification) are used in both the training phase and the recognition phase.

3.4.1 Keystroke Feature Extraction

3.4.1.1 Keystroke Extraction

Typical users can type up to about 300 characters per minute. Keystrokes consist of a push and a release. Our experiments confirm Asonov and Agrawal's observation that the period from push to release is typically about 100 milliseconds. There is usually more than 100 milliseconds between consecutive keystrokes, which is large enough to distinguish the consecutive keystrokes. Figure 3.3 shows the acoustic signal of a push peak and a release peak. We need to detect the start of a keystroke, which is

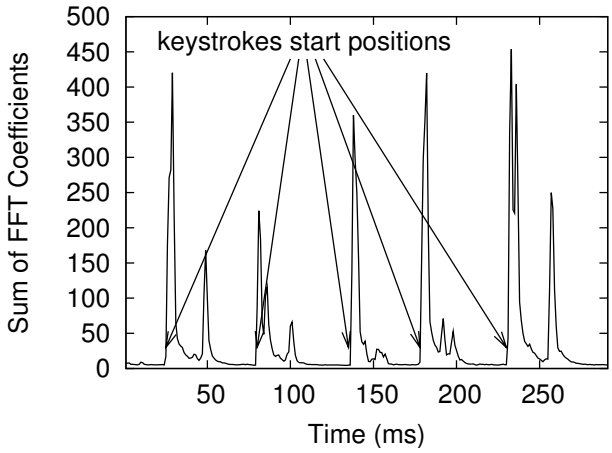

Figure 3.4: Energy levels over the duration of 5 keystrokes. (Smaller peaks are release peaks.)

essentially the start of the push peak in a keystroke acoustic signal.

We distinguish between keystrokes and silence using energy levels in time windows. In particular, we calculate the windowed discrete Fourier transform of the signal and use the sum of all FFT coefficients as energy. We use a threshold to detect the start of keystrokes. Figure 3.4 shows an example.

3.4.1.2 Features: Cepstrum vs. FFT

Given the start of each keystroke (`wav_position`), features of this keystroke are extracted from the audio signal during the period from `wav_position` to `wav_position`+ ΔT. Our experiments compare two different types of features. First we use FFT features with $\Delta T \approx 5$ms, as in [Asonov and Agrawal, 2004]. This time period roughly corresponds to the *touch peak* of the keystroke, which is when the finger touches the key. An alternative is to use the *hit peak*, when the key hits the supporting plate. The hit peak is harder to pinpoint in the signal, so our experiments use the *touch peak*.

As shown in Figure 3.1, the classification results using FFT features are not sat-

isfactory and we can not achieve the levels reported in [Asonov and Agrawal, 2004]. This might be caused by different experimental environment settings, different quality of recording devices, etc.

Next, we use cepstrum features. Cepstrum features are widely use in speech analysis and recognition [Childers *et al.*, 1977]. Cepstrum features have been empirically verified to be more effective than plain FFT coefficients for voice signals. In particular, we use Mel-Frequency Cepstral Coefficients (MFCCs) [Jurafsky and Martin, 2000]. In our experiments, we set the number of channels in the Mel-Scale Filter Bank to 32 and use the first 16 MFCCs computed using 10ms windows, shifting 2.5ms each time. MFCCs of a keystroke are extracted from the period from `wav_position` to `wav_position` $+ \Delta T'$, where $\Delta T' \approx$ 40ms which covers the whole push peak. As Figure 3.1 reports, this yields far better results than from FFT features.

Asonov and Agrawal's observation shows that high frequency acoustic data provides limited value. We ignore data over 12KHz. After feature extraction, each keystroke is represented as a vector of features (FFT coefficients or MFCCs).

3.4.2 Unsupervised Single Keystroke Recognition

As discussed above, the unsupervised recognition step recognizes keystrokes using audio recording data only and no training or language data.

The first step is to cluster the feature vectors into K acoustic classes. Possible algorithms to do this include K-means and Expectation-Maximization (EM) on Gaussian mixtures [Bilmes, 1997]. Our experiments tested values of K from 40 to 55, and $K = 50$ yielded the best results. We use thirty keys, so K must be equal or larger than 30. A larger K captures more information from the sound samples, but it also makes the system more sensitive to noise. It would be interesting to experiment with using Dirichlet processes that might predict K automatically [Jordan, 2008].

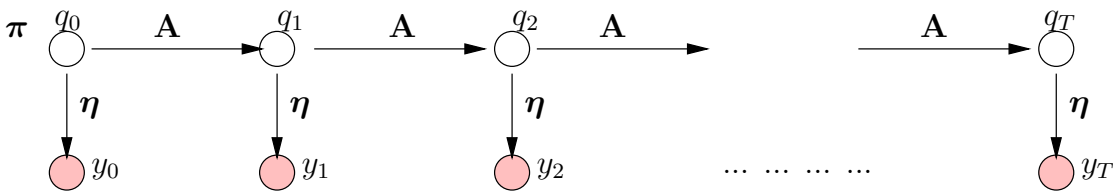

Figure 3.5: The Hidden Markov Model for unsupervised key recognition

The second step is to recover text from these classes. For this we use a Hidden Markov Model (HMM) [Rabiner and Juang, 1986]. HMMs are often used to model finite-state stochastic processes. In a Markov chain, the next state depends only on the current state. Examples of processes that are close to Markov chains include sequences of words in a sentence, weather patterns, etc. For processes modeled with HMM, the true *state* of the system is unknown and thus is represented with *hidden* random variables. What is known are *observations* that depend on the state. These are represented with *known* output variables. One common problem of interest in an HMM is the *inference problem*, where the unknown state variables are inferred from a sequence of observations. This is often solved with the Viterbi algorithm [Russell and Norvig, 2003]. Another problem is the *parameter estimation problem*, where the parameters of the conditional distribution of the observations are estimated from the sequence of observations. This can be solved with the EM algorithm.

Figure 3.5 shows the HMM we used. It is represented as a statistical graphical model [Jordan, 2008][2]. Circles represent random variables. Shaded circles (y_i) are observations while unshaded circles (q_i) are unknown state variables we wish to infer. Here, q_i is the label of the i-th key in the sequence, and y_i is the class of the keystroke we obtained in the clustering step. The arrows from q_i to q_{i+1} and from q_i to y_i indicate that the latter is conditionally dependent on the former; the value on the arrow is an

[2]One might think that a more generalized Hidden Markov Model, such as one that uses Gaussian mixture emissions [Jordan, 2008], would give better results. However, the HMM with Gaussian mixture emission has a much larger number of parameters and thus faces the "overfitting" problem. We find a discrete HMM as presented here gave better results.

entry in the probability matrix. So here we have $p(q_{i+1}|q_i) = A_{q_i,q_{i+1}}$, which is the probability of the key q_{i+1} appearing after key q_i. The A matrix is another way of representing plaintext bigram distribution data. The A matrix (called the transition matrix) is determined by the English language and thus is obtained from a large corpus of English text. We also have $p(y_i|q_i) = \eta_{q_i,y_i}$, which is the probability of the key q_i being clustered into acoustic class y_i in the previous step. Our observations (the y_i values) are known. The output matrix η is unknown. We wish to infer the q_i values. Note that one set of values for q_i and η are better than a second set if the likelihood (joint probability) of the whole set of variables, computed by multiplying all conditional probabilities, is larger with the first set than the second set. Ideally, we want a set of values that maximize the likelihood, so we are performing a type of Maximum Likelihood Estimation [Russell and Norvig, 2003].

We use the EM algorithm [Bilmes, 1997] for parameter estimation. It goes through a number of rounds, alternately improving q_i and η. The output of this step is the η matrix. After that, the Viterbi algorithm [Russell and Norvig, 2003] is used to infer q_i, i.e. the best sequence of keys.

EM is a randomized algorithm. Good initial values make the chance of getting satisfactory results better. We find initializing the row in η corresponding to the Space key to an informed guess makes the EM results more stable. This is probably because spaces delimit words and strongly affect the distribution of keys before and after the spaces. This task is performed manually. Space keys are easy to distinguish by ear in the recording because of the key's distinctive sound and frequency of use. We mark several dozen space keys, look at the class that the clustering algorithm assigns to each of them, calculate their estimated probabilities for class membership, and put these into η. This approach yields good results for most of the runs. However, it is not necessary. Even without knowing where Space keys occur, EM with different random initial values eventually yields a good set of parameters. All other keys used

in our study, including punctuation keys are initialized to random values in η. We believe that initialization of η can be completely automated in the future.

3.4.3 Error Correction with a Language Model

As we discuss in Section 3.3, error correction is a crucial step in improving the results. It is used in unsupervised training, supervised training and also recognition of English text.

3.4.3.1 Simple Probabilistic Spelling Correction

Using a spelling checker is one of the easiest ways to exploit knowledge about the language. We ran spell checks using *Aspell* [Atkinson, 2005a] on recognized text and found some improvements. However stock spell checkers are limited in the kinds of spelling errors they can handle, e.g. at most two letters wrong in a word. They are designed to cope well with common errors that human typists make, not the kinds of errors that acoustic keystroke classifiers make. It is not surprising that their utility here is limited.

Fortunately, there are patterns in the errors that the acoustic keystroke classifier makes. For example, it may have difficulty with several keys, often confusing one with another. Suppose that we know the correct plaintext. (This is of course not true, but as we iterate the algorithm, we predict the correct plaintext with increasing accuracy. Below, we address the case of unsupervised step, where we know no plaintext at all.) Under this assumption, we would have a simple method to exploit these patterns. We act as if this assumption were true, and run the acoustic keystroke classifier on training data and record all classification results, including errors. With this, we calculate a matrix E (sometimes called the confusion matrix in the machine learning

literature),

$$E_{ij} = \hat{p}(y = i|x = j) = \frac{N_{x=j,y=i}}{N_{x=j}} \tag{3.1}$$

where $\hat{p}(\cdot)$ denotes estimated probability, x is the typed key and y is the recognized key, and $N_{x=j,y=i}$ is the number of times $x = j, y = i$ is observed. Columns of E give the estimated conditional probability distribution of y given x.

Assume that letters are independent of each other and the same is true for words. (This is a false assumption because there is significant inter-letter dependence in natural languages, but works well in practice for our experiments.) We compute the conditional probability of the recognized word $\mathbf{Y}$ (the corresponding string returned by the recognizer, not necessarily a correct word) given each dictionary word $\mathbf{X}$.

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{\text{length of } \mathbf{X}} p(\mathbf{Y}_i|\mathbf{X}_i) \approx \prod_i E_{y_i,x_i} \tag{3.2}$$

In the equation above, $\mathbf{X}_i$ is the i-th character of dictionary word $\mathbf{X}$ and $\mathbf{Y}_i$ is the i-th character of the recognized word. $p(\mathbf{Y}|\mathbf{X})$ represents the probability that the recognition result is Y but the actual user input word is X.

We compute this probability for each dictionary word, which takes only a fraction of a second. The word list we use is SCOWL [Atkinson, 2005b] which ranks words by complexity. We use words up to level 10 (higher-level words are more obscure), which covers most commonly used words, giving us 95,997 words in total. By simply selecting the word with the largest posterior probability as our correction result, we correct many errors.

Because of the limited amount of training data, there will be many zeroes in E if Equation (3.1) is used directly, that is, the matrix will be sparse. This is undesirable because the corresponding combination may actually occur in the recognition data. This problem is similar to the zero-occurrence problem in n-gram models [Jurafsky

and Martin, 2000]. We assign an artificial occurrence count (we use 0.1) to each zero-occurrence event.

In the discussion above we assume the plaintext is known, but we do not even have an approximate idea of the plaintext in the first round of (unsupervised) training. We work around this by letting $E_{ii} = p_0$ where p_0 is a constant (we use 0.5) and distribute the remaining $1 - p_0$ uniformly over all E_{ij} where $j \neq i$. Obviously this gives suboptimal results, but the feedback mechanism corrects this later.

3.4.3.2 Adding an n-gram Language Model

The spelling correction scheme above does not take into account relative word frequency or grammar issues: for example, some words are more common than others, and there are rules in forming phrases and sentences. Spelling correction will happily accept "fur example" as a correct spelling because "fur" is a dictionary word, even though the original phrase is probably "for example".

One way to fix this is to use an n-gram language model that models word frequency and relationship between adjacent words probabilistically [Jurafsky and Martin, 2000]. Specifically, we combine trigrams with the spelling correction method above and model a sentence using the graphical model shown in Figure 3.6. The hidden variables w_t are words in the original sentence. The observations v_t are recognized words. $p(v_t|w_t)$ is calculated using Equation (3.2) above. Note this is a second-order HMM, because every hidden variable depends on two prior variables. The conditional probability $p(w_t|w_{t-1}, w_{t-2})$ is determined by a trigram model obtained by training on a large corpus of English text.

In this model only the w_i values are unknown. To infer the most likely sentence, we again use the Viterbi algorithm. We use a version of the Viterbi algorithm for second order HMMs, similar to the one in [Thede and Harper, 1999]. The complexity of the algorithm is $O(TN^3)$, where T is the length of the sentence and N is the number

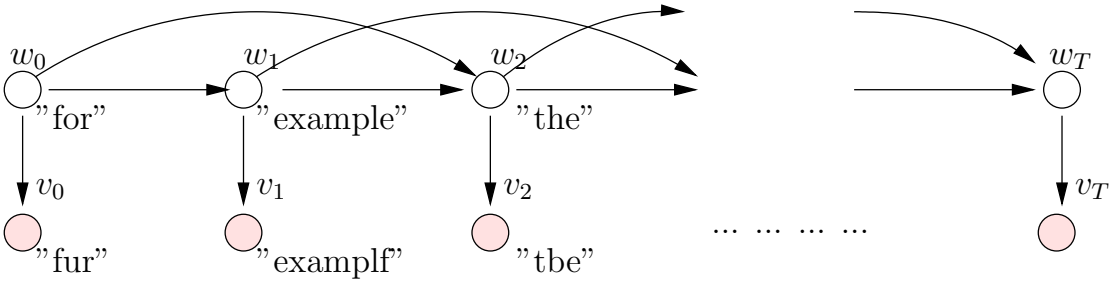

Figure 3.6: Trigram language model with spell correction

of possible values for each hidden variable, that is, the number of dictionary words of the appropriate length. To reduce complexity, only the top M candidates from the spelling correction process of each word are considered in the Viterbi algorithm, lowering the cost to $O(TM^3)$. That is, for each recognized word v_t, we select the top M possible hidden variables (w_t) where $p(v_t|w_t)$ are the largest values among all dictionary words. We start from the first word, and each word is chosen from the top M candidate dictionary words. We find the path with the largest:

$$p(v_1|w_1)p(w_2|w_1)p(v_2|w_2)\prod_{t=3}^{T} p(v_t|w_t)p(w_t|w_{t-1}, w_{t-2})$$

We used $M = 20$ in our experiments. Larger M values provide little improvement.

3.4.4 Supervised Training and Recognition

Supervised training refers to training processes performed with labeled training data. We apply our feedback-based training processes iteratively, using in each iteration characters "recognized" in previous iterations as training samples to improve the accuracy of the acoustic keystroke classifier.

Below, we discuss three different methods of supervised training and recognition we use in our experiments, including the one used in [Asonov and Agrawal, 2004].

Like any supervised classification problem, there are two stages:

1. Training: input feature vectors and corresponding labels (the key pressed) and output a model to be used in recognition;

2. Recognition: input feature vectors and the trained classification model and output the label of each feature vector (keystroke).

3.4.4.1 Method 1: Neural Networks

The first method is neural networks, also used by Asonov and Agrawal [Asonov and Agrawal, 2004]. Specifically, we use probabilistic neural networks, which are arguably the best neural networks available for for classification problems [Wasserman, 1993]. We use Matlab's `newpnn()` function, with spread radius parameter as 1.4 (this gives the best results in our experiments).

3.4.4.2 Method 2: Linear Classification (Discriminant)

The second method is simple linear (discriminant) classification [Jordan, 2008]. This method assumes the data to be Gaussian and finds hyperplanes in the space to divide the classes. We use the `classify()` function from Matlab.

3.4.4.3 Method 3: Gaussian Mixtures

The third method is more sophisticated than linear classification (though it gave worse results in our experiments). Instead of assuming Gaussian distribution of data, it assumes that each class corresponds to a *mixture* of Gaussian distributions [Jordan, 2008]. A *mixture* is a distribution composed of several sub-distributions. For example, a random variable with distribution of a mixture of two Gaussians could have a probability of 0.6 of being in one Gaussian distribution and 0.4 of being in the other

	recording length	number of words	number of keys
Set 1	12m17s	409	2514
Set 2	26m56s	1000	5476
Set 3	21m49s	753	4188
Set 4	23m54s	732	4300

Table 3.1: Statistics of each test set

Gaussian distribution. This captures the fact that each key may have several slightly different sounds depending on how the typist hit the key.

We also use the EM algorithm to train the Gaussian mixture model. In our experiment, we used mixtures of five Gaussian distributions of diagonal covariance matrices. Mixtures of more Gaussians provide potentially better model accuracy but need more parameters to be trained, requiring more training data and often making EM less stable. We find using five components seems to provide a good tradeoff. Using diagonal covariance matrices reduces the number of parameters. Without this restriction, EM has very little chance of yielding a useful set of parameters.

3.5 Evaluation

Our experiments evaluated the attacks. In our first experiment, we worked with four recordings of various lengths of news articles being typed. We used a Logitech Elite cordless keyboard in use for about two years (manufacturer part number: 867223-0100), a $10 generic PC microphone and a Soundblaster Audigy 2 soundcard. The typist was the same for each recording. The keys typed included “a”-“z”, comma, period, space and enter. The article was typed entirely in lower case so the shift key was never used. Typists were told to continue typing without using backspace key for error correction. (We discuss these issues in Section 3.6.)

Table 3.1 shows the statistics of each test set. Sets 1 and 2 are from quiet environments, while sets 3 and 4 are from noisy environments. Our algorithm for detecting

the start of a keystroke sometime fails. We manually corrected the results of the algorithm for sets 1, 2 and 3, requiring ten to twenty minutes of human time per data set. (Sets 1 and 2 needed about 10 corrections; set 3 required about 20 corrections.) For comparison purposes, set 4 (which has about 50 errors in determining the start of keystrokes) was not corrected.

In our second experiment, we recorded keystrokes from three additional models of keyboards (see Section 3.5.1.2). The same keystroke recognition experiments were run on these recordings and results compared. We used identical texts in this experiments on all these keyboards.

3.5.1 English Text Recognition

3.5.1.1 A Single Keyboard

In our experiments, we used linear classification to train the keystroke classifier. Table 3.2 shows the result after each step. First, the unsupervised learning step (Figure 3.2(a)) was run. In this unsupervised step, the HMM model shown in Figure 3.5 was trained using EM algorithm described above[3]. The output from this step is the recovered text from HMM/Viterbi unsupervised learning, and the text after language model correction. These two are denoted as *keystrokes* and *language* respectively in the table. Then the first round of feedback supervised training produces a new classifier. The iterated corrected text from this classifier (and corresponding text corrected by the language model) are shown in the row marked "1st supervised feedback". We perform three rounds of feedback supervised learning. The bold numbers show our final results. The bold numbers in the "language" row are the final recognition rate we achieve for each test set. The bold numbers in the "keystroke" row are the recog-

[3]Since the EM algorithm is a randomized algorithm, it might sometimes get stuck in local optima. To avoid this, in each of these experiments we run the same training process eight times and used results from the run with the highest log-likelihood.

nition rates of the keystroke classifier, without using the language model. These are the recognition rates for random or non-English text.

The results show that:

- The language model correction greatly improved the correct recovery rate for words.
- The recovery rates in quiet environments (sets 1 and 2) were slightly better that those in noisy environments (sets 3 and 4). But the difference became smaller after several rounds of feedback.
- Correctness of the keystroke position detection affected the results. The recovery rate in set 3 was better than set 4 because of keystroke location mistakes included in set 4.
- When keystroke positions have been corrected after several rounds of feedback, we achieved an average recovery rate of 87.6% for words and 95.7% for characters.

		Set 1		*Set 2*		*Set 3*		*Set 4*	
		words	chars	words	chars	words	chars	words	chars
unsupervised learning	keystrokes	34.72	76.17	38.50	79.60	31.61	72.99	23.22	67.67
	language	74.57	87.19	71.30	87.05	56.57	80.37	51.23	75.07
1st supervised feedback	keystrokes	58.19	89.02	58.20	89.86	51.53	87.37	37.84	82.02
	language	89.73	95.94	88.10	95.64	78.75	92.55	73.22	88.60
2nd supervised feedback	keystrokes	65.28	91.81	62.80	91.07	61.75	90.76	45.36	85.98
	language	90.95	96.46	88.70	95.93	82.74	94.48	78.42	91.49
3rd supervised feedback	keystrokes	66.01	**92.04**	62.70	**91.20**	63.35	**91.21**	48.22	**86.58**
	language	**90.46**	**96.34**	**89.30**	**96.09**	**83.13**	**94.72**	**79.51**	**92.49**

Table 3.2: Text recovery rate at each step. All numbers are percentages. The outputs denoted as "keystroke" are recovery rates before language model correction. The bold face numbers in the "keystroke" row represent recovery rates that could be achieved for random sequences of characters. The outputs denoted as "language" are recovery rates after language model correction. The bold face numbers in the "language" row represent recovery rates that could be achieved for non-random sequences of characters, such as English text.

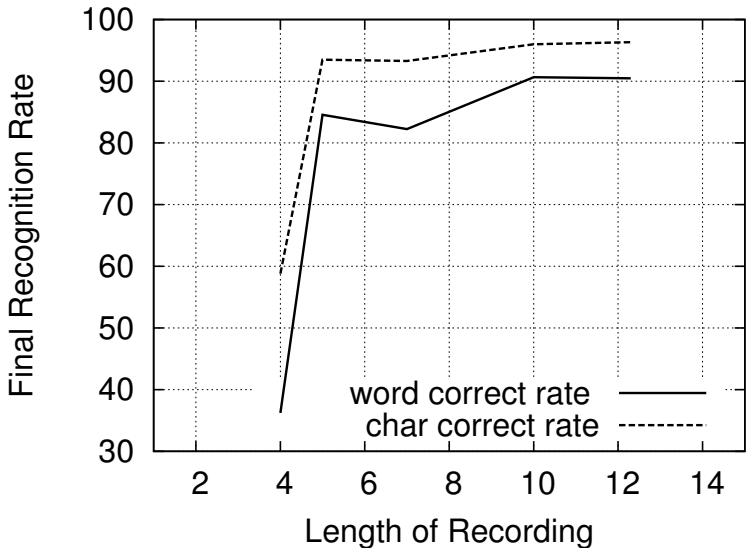

Figure 3.7: Length of recording vs. recognition rate

To understand how different classification methods in the supervised training step affected the results, we reran the same experiment on set 1, using different supervised classification methods. Table 3.3 shows our results. The methods in order of quality are is linear classification, then Gaussian mixtures, and then neural networks. Experiments with other data sets gave similar results.

In the experiments above, we used recordings longer than 10 minutes. To discover the minimal amount of training data needed for reasonable results, we took the first data set (i.e. "Set 1" above) and used only the first 4, 5, 7 and 10 minutes of the 12-minute recording for training and recognition. Figure 3.7 shows the recognition results we get. This figure suggests that at least 5 minutes of recording data are sufficient to get good results for this particular recording[4].

[4]The dip in the solid curve probably occurred because of noise during the 2-minute recording window (between minute 5 and minute 7).

		NN		*LC*		*MC*	
		words	chars	words	chars	words	chars
1st supervised feedback	keystrokes	59.17	87.07	58.19	89.02	59.66	87.03
	language	80.20	90.85	89.73	95.94	78.97	90.45
2nd supervised feedback	keystrokes	70.42	90.33	65.28	91.81	66.99	90.25
	language	81.17	91.21	90.95	96.46	80.20	90.73
3rd supervised feedback	keystrokes	71.39	**90.81**	66.01	**92.04**	69.68	**91.57**
	language	**81.42**	**91.93**	**90.46**	**96.34**	**83.86**	**93.60**

Table 3.3: Recognition rates of classification methods in supervised learning. All numbers are percentages. The outputs denoted as "keystroke" are recovery rates before language model correction. The bold face numbers in the "keystroke" row represent recovery rates that could be achieved for random sequences of characters. The outputs denoted as "language" are recovery rates after language model correction. The bold face numbers in the "language" row represent recovery rates that could be achieved for non-random sequences of characters, such as English text. (NN:Neural Network; LC:Linear Classification; MC:Gaussian Mixtures)

3.5.1.2 Multiple Keyboards

To verify that our approach applies to different models of keyboards, we performed the keystroke recognition experiment on different keyboards, using linear classification in the supervised training step. The models of the keyboards we used are:

- Keyboard 1: Dell Quietkey PS/2 keyboard, manufacturer part number 2P121, in use for about 6 months.
- Keyboard 2: Dell Quietkey PS/2 keyboard, manufacturer part number 035KKW, in use for more than 5 years.
- Keyboard 3: Dell Wireless keyboard, manufacturer part number W0147, new.

The same document (2273 characters) was typed on all three keyboards and we recorded keystroke sounds. Each recording lasted about 12 minutes. In these recordings, the background machine fan noise was noticeable. While recording from the third keyboard, we got several seconds of unexpected noise from a cellphone nearby.

		Keyboard 1		*Keyboard 2*		*Keyboard 3*	
		words	chars	words	chars	words	chars
unsupervised learning	keystrokes	30.99	71.67	20.05	62.40	22.77	63.71
	language	61.50	80.04	47.66	73.09	49.21	72.63
1st supervised feedback	keystrokes	44.37	84.16	34.90	76.42	33.51	75.04
	language	73.00	89.57	66.41	85.22	63.61	81.24
2nd supervised feedback	keystrokes	56.34	88.66	54.69	86.94	42.15	81.59
	language	80.28	92.97	76.56	91.78	70.42	86.12
Final result	keystrokes	60.09	**89.85**	61.72	**90.24**	51.05	**86.16**
	language	**82.63**	**93.56**	**82.29**	**94.42**	**74.87**	**89.81**

Table 3.4: Text recovery rate at each step. With different keyboards. All numbers are percentages. The outputs denoted as "keystroke" are recovery rates before language model correction. The bold face numbers in the "keystroke" row represent recovery rates that could be achieved for random sequences of characters. The outputs denoted as "language" are recovery rates after language model correction. The bold face numbers in the "language" row represent recovery rates that could be achieved for non-random sequences of characters, such as English text.

Table 3.4 shows our results. Results in the table show that the first and the second keyboards achieve higher recognition rate than the third one. However, all keyboards we tested are vulnerable to the attacks we present in this paper.

3.5.2 Random Text Recognition and Password Stealing

We used the keystroke classifier trained by set 1 to mount password stealing attacks. All password input recorded in our experiment were randomly generated sequences, not user names or dictionary words. The output of the keystroke classifier for each keystroke is a set of posterior probabilities:

$$p(\text{this keystroke has label } i|\text{observed-sound}), \quad i = 1, 2, \ldots, 30.$$

Given these conditional probabilities, one can calculate probabilities for all sequences of keys being the real password. We sorted these sequences by their probabilities

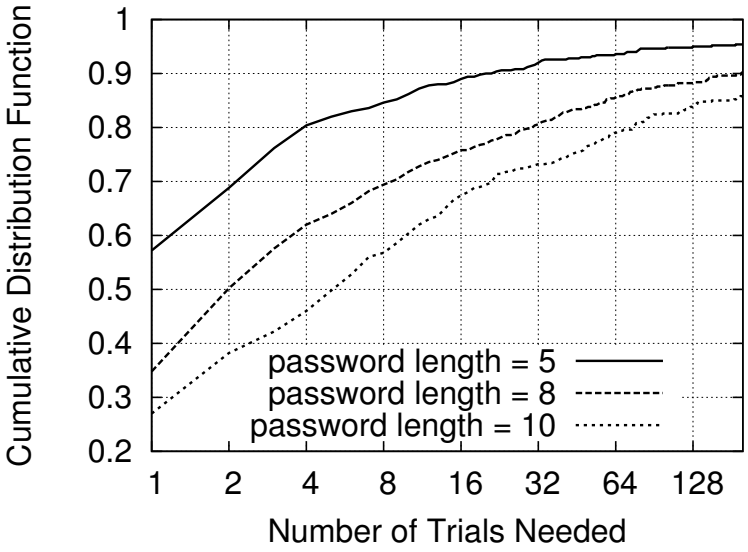

Figure 3.8: Password stealing: distribution of the number of trials required by the attacker.

from the largest to the smallest. This produced a candidate list and the attacker can try one-by-one from the top to the bottom. To measure the efficacy of the attack, we used the position of the real password in this list. A user inputed 500 random passwords each of length 5, 8 and 10. Figure 3.8 shows the cumulative distribution function of the position of the real password. For example, with twenty trials, 90% of 5-character passwords, 77% of 8-character passwords and 69% of 10-character passwords are recovered. As Figure 3.8 also shows, after seventy-five trials, we can recover 80% of 10-character passwords.

3.6 Discussion

3.6.1 Timing Information

We discuss above how to use acoustic information of keystrokes to recover typed keys. Our experiments show high keystroke recovery rates using only acoustic information.

As mentioned above, Song, Wagner and Tian point out that the time between consecutive keys also carries information about typed keys [Song *et al.*, 2001]. It may be possible to further improve the recovery rate with timing information. Here we give one way to combine timing features with acoustic features, however our results show only a modest improvement in recovery rate.

The time between a pair of consecutive keys is related to many factors, such as the location of the two keys on the keyboard, typing style, whether the keys are typed by alternating hands or the same hands, whether the keys are typed by different fingers or the same finger, etc. We recorded a typist at normal pace, without intentional stops between keys. Figure 3.9 shows the distribution of time between "a" and subsequent keys is significantly different from "h" and subsequent keys. The key "a" is located near the border of a keyboard and touch typists use the small finger of the left hand to type it; while the key "h" is located in the middle of a keyboard and touch typists use the index finger of the right hand to type it. Figure 3.9 suggests the time between a key and a subsequent key carries information about the location of the key on the keyboard, that is, information related to the label of the key.

In the discussion above we represent acoustic information as a vector of features. If we assume the length of the time interval between a key and its next key carries information (as shown in Figure 3.9), we can add time as an additional dimension in the feature vector. We can then apply new feature vectors with time as one of the dimensions in our supervised training step. We experimented with the sets 1, 2, and 3, using training approaches as above. Table 3.5 shows that initial and final recognition rates in supervised training.

The recognition rates in Table 3.5 suggest that time between consecutive keys does not substantially improve the supervised learning in the feedback based training. The results are not as good as the results reported by Song, Wagner and Tian. Reasons for this discrepancy may include:

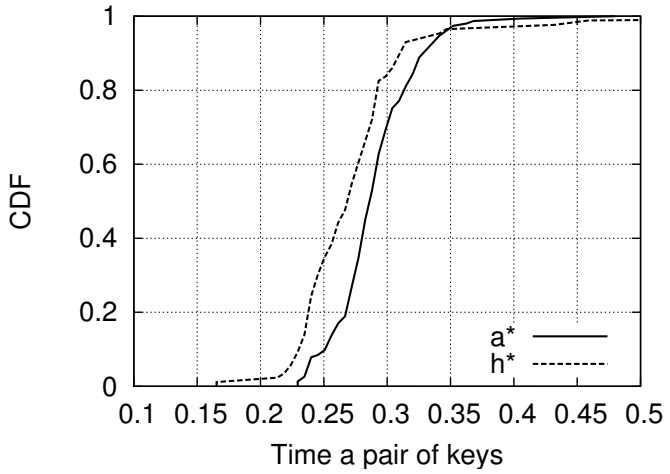

Figure 3.9: Cumulative distribution function (CDF) of time: a pair of keys starting with "a" ("a*") vs. a pair of keys starting with "h" ("h*").

- Song et al. used the length of the time interval between consecutive keys in a very short phase, such as a password. The pace of typing when a user types his password is probably more consistent than the pace of typing when a user types an article. In our test sets, the typist sometimes stopped in the middle of typing an article. Also typing speed for some words are much faster than others even if those words share common pairs of characters. The newly introduced timing information ("signal") comes along with random variations ("noise") above. When we add the timing information with acoustic information, the training methods do not receive a sufficient number of samples with consistent timing information to improve recovery rates.

- Acoustic information alone yields a high recognition rate. The acoustic information has a higher "signal-to-noise" ratio compared to timing information. Moreover, spelling and grammar correction makes the effects of timing information less visible too.

Finding good ways to combine inter-keystroke time interval information with

			without time		with time	
			word	char	word	char
set 1	initial	keystroke	58.19	89.02	57.95	88.94
		language	89.73	95.94	89.00	95.31
	final	keystroke	66.01	92.04	65.77	92.12
		language	90.46	96.34	**91.20**	**96.50**
set 2	initial	keystroke	58.20	89.86	58.20	89.83
		language	88.10	95.64	87.7	95.28
	final	keystroke	62.70	91.20	62.70	91.09
		language	89.30	96.09	89.2	96.00
set 3	initial	keystroke	51.53	87.37	51.39	87.32
		language	78.75	92.55	77.56	92.09
	final	keystroke	63.35	91.21	62.55	91.07
		language	83.13	94.72	82.20	94.44

Table 3.5: Recognition rate in supervised training: with timing information vs. without timing information

acoustic key recovery merits further research.

3.6.2 Why Keys Sound Different

We are interested in finding out why keystrokes of different keys sound different. There are at least two contributing factors:

- *Keyboard layout.* Keys are located at different locations on the support plate of a keyboard. Just as striking a drum at different locations yields different sounds, the keys on a keyplate yield different sounds.

- *Typing patterns.* The sound of keystroke is related to how the key is typed; for example, the direction that a key is hit.

To verify the hypotheses above, we performed an experiment.

In the experiment, each key was repeatedly struck 50 times. The sound samples of 50 hits for each key were used to train acoustic classifiers using different classification

methods (i.e. linear classification, neural networks and mixture of Gaussians). Then, the acoustic classifiers trained in this way were used to recognize the training set and two different sets of new sound samples. The first test set was composed of sound samples from 30 repeated hits of each key. The second test set is composed of sound samples from a typist typing an article. Table 3.6 shows our recognition rates.

	repeat50 (training)	*repeat30*	*article*
Linear Classification	*95.67%*	88.05%	53.49%
Neural Network	*100%*	81.84%	51.21%
Mixture of Gaussians	*98.87%*	81.15%	47.44%

Table 3.6: Recognition rate of repeat key hits and article input using classifier trained by repeat key hits

When a key was repeatedly struck, all keystrokes are made with the similar strength by a single finger and using almost the same gesture. There is no difference in typing style between different keys. Table 3.6 shows that the recognition rates of test samples from keys typed repeatedly are over 80%, which suggests that the sound differences may come from the physical properties of a keyboard: location of keys, physical difference between keys, etc. This observation supports our first assumption that keyboard layout contributes to different sound from keys.

If the keyboard layout were the only reason for different sound of keys, the classifier trained by sound of repeatedly typing the same key should also work for normal typing. However, this model was not very effective in classifying normal English text, because typing normal English text uses a variety of paces, gestures, and key press strengths. Repeated typing only teaches the acoustic classifiers a portion of sounds that a key could make. This experiment suggests that keys sound different because of both the keyboard layout and typing style such as paces, gestures and hitting strengths, etc.

3.6.3 Special Keys

The current attack does not take into account special keys such as the Shift key, the Control key, the Backspace key and the Caps Lock key. There are two issues here. One is whether keystrokes of special keys are separable from other keystrokes at signal processing time. Our preliminary experiments suggest this is possible; push peaks of keystrokes are easily separable in the recordings we looked at. The other issue is how modifier keys such as the Shift key fit into spelling correction scheme. We hypothesize ad hoc solutions such as replacing the Shift key or the Caps Lock key with the Space key will work. The Shift key often appears before the first letter of a word. If it is recognized incorrectly, the following word will be one letter longer. In spelling and grammar correction, we can take this into account by not only considering words of the same lengths, but also those with one fewer letter. For example, if we get "atje" after initial recognition, the word "the" will also be considered as a candidate word for correction because we might misrecognize the Shift key as "a". These keys can be much more reliably recognized by training a classifier specifically for the Shift key and the Caps Lock key. Note that we do not need to distinguish between uppercase and lowercase in the recovered text, so it is not necessary to detect when the Shift key is released.

The Backspace key is also important. The ideal solution would be to discover what the final text is after applying the backspaces. But that complicates error correction algorithms. So one can just recognize these keys and leave the "word" before and after out of error-correction because they are probably not full words. An interesting fact about the Backspace key is that this key is sometimes struck repeatedly: a user sometimes wants to delete a whole word or a whole sentence. Here, a bit of human aid could be useful because the Backspace key is relatively easy to detect by ear based on sound and context, although it is harder to detect than the Space key. It

is not difficult for human ears to detect repeated keystrokes of the Backspace key. Since the sound of the Backspace key is very different from others, in the acoustic clustering step, they will normally clustered with the same label. It is possible to write a program to automatically select sound samples of consecutive keys which are clustered in a common label. A variety of techniques could be used to decide whether these are consecutive Backspaces. After sound samples of the Backspace keys are collected, we train a specific acoustic classifier for the Backspace keys as well.

3.6.4 Attack Improvements

This section discusses topics for future research that could improve our attack:

- One challenge we met in our work was marking keystroke starting points in the sound signal. This is not trivial because the sound varies in energy level, timing, frequency distribution, etc., depending on the typist and recording environment. We use energy level and timing constraints between consecutive keys to mark the starting positions of keystrokes. Detection rules are manually created based on past experiences. Our detection program based on this approach has difficulty in marking keystroke positions in recordings from fast typists. However, there is additional information we can use: namely frequency, which appears to vary from the push peak to the release peak. This is a topic for future research. A robust and consistent keystroke position detection algorithm may also improve the recovery rate of typed characters.

- Currently, we assume the Space key, Enter key and punctuation keys are detected correctly and use them to divide characters into words. We use candidate words of the same length as the "words" separated in this way. A topic for future research is to explore better ways to choose candidate words for correction, with

the goal of high quality correction even when there are mistakes in separating words.

- An alternative method for feedback training is Hierarchical Hidden Markov Models (HHMMs) [Fine *et al.*, 1998]. In a HHMM, HMMs of multiple levels (grammar level and spelling level in this case) are built into a single model. Algorithms to maximize global joint probability may improve the effectiveness of the feedback training procedure. This approach merits further investigation.

- Our experiments tested on FFT features and cepstrum features. However, there are other types of features for representing sound signals. For each type of feature, there are multiple parameters to control the extracted information. Currently, we used ad hoc methods to select these parameters. An entropy based metric defined specifically for measuring acoustic features may provide better, more systematic way to compare features and parameters. This metric may also allow us to compare information leaked by individual keys. Given current PC keyboard layouts, is the leaking uniform among keys, or should we pay more attention to specific keys? Is it possible to know which typing styles leak more information and whether different typists leak different amounts of information?

- In a controlled environment where we can record isolated typing sounds, the recovery rate is now high. However, in most realistic situations, environmental background noise is an issue. In many work spaces, we have multiple users simultaneously typing. Isolating the sound of a single typist is difficult. It is interesting to consider recording with multiple microphones, including arrays of directional microphones. We could get the sound signal of multiple channels in this way. Similarly, we have shown that the recognition rate is lower in noisy environments. Attacks will be less successful when the user is playing music or

talking to others while typing. However, we may be able to use signal processing techniques (especially in multichannel recordings) to isolate the sound of a single typist.

- We hope to explore a variety of recording devices including parabolic microphones, laser microphones, telephone receiver microphones, acoustic chat connections such as Skype, etc.

- In future work, it is particularly interesting to try to detect keystrokes typed in a particular application, such as a visual editor (e.g. emacs) or a software development environment (e.g. Eclipse). Examining text typed in these environment presents challenges because more keys maybe used and special keys maybe used more often. Furthermore, the bigram or transition matrix will be different. Nonetheless we believe that our techniques may be applicable to detecting keystrokes of users in these applications and indeed can even cover input as different as other small alphabet languages, such as Russian or Arabic, large alphabet languages, such as Chinese or Japanese, and even programming languages.

3.6.5 Defenses

Since our attack is based on acoustic signals derived from passively eavesdropping, it is more difficult to detect this type of attacks than through active attacks where attackers interact with victims. Here are some preliminary ideas about potential defenses:

- One can reduce the possibility of leaking acoustic signals. Sound-proofing may help, but given the effectiveness of modern parabolic and laser microphones, the standards are high.

- Quieter keyboards as suggested by Asonov and Agrawal may reduce vulnerability. However, the two so-called "quiet" keyboards we used in our experiments proved ineffective against the attack. Asonov and Agrawal also suggest that keyboard makers could produce keyboards having keys that sound so similar that they are not easily distinguishable. They claim that one reason keys sound different today is that the plate underneath the keys makes different sounds when hit at different places. If this is true, using a more uniform plate may alleviate the attack. However, it is not clear whether these kinds of keyboards are commercially viable. Also, there is the possibility that more subtle differences between keys can still be captured by an attacker. Further, keyboards may develop distinct keystroke sounds after months of use.

- Another approach is reduce the quality of acoustic signal that can be acquired by attackers. We can add masking noise while typing. However, we are not sure that masking noises might not be easily separable. As we discussed above, an array of directional microphones may be able to record and distinguish sound into multiple channels according to the locations of the sound sources. This defense will be less effective when attackers are able to collect more data. Masking has another problem: it annoys typists. Perhaps a short window of noise could be added at every predicted push peak. This may be more acceptable to typists than continuous masking noise. Alternatively, perhaps we could randomly insert noise windows which sound like push peaks of keystrokes.

- The practice of relying only on typed passwords or even long passphrases should be reexamined. One alternative is two-factor authentication that combines passwords or pass-phrases with smart cards, one-time-password tokens, biometric authentication and etc. However two-factor authentication does not solve all our problems. Typed text other than passwords is also valuable to attackers.

Chapter 4

Side Channel System Defense: Detecting Botnets from Spam Email Messages

4.1 The Case: Identifying Botnets from Spam Email Messages

In this chapter, we study a type of side channel trace created by attackers, spam email messages received through the Hotmail Web mail service, to discover information about collections of machines (botnets) that perform coordinated malicious acts. We use information logged in headers of email about the sender, receiver and the relay servers. This information is logged by the SMTP (Simple Mail Transfer Protocol) widely used for email.

In recent years, malware has become a widespread problem. As a result, malicious users or organizations have been able to gain remote control of an increasingly large number of machines. Once compromised, these machines run software that accepts

and executes commands from the controller. These compromised machines are generally referred to as *bots*, and the set of bots controlled by a single entity is called a *botnet*. Botnet controllers use techniques such as IRC channels and customized peer-to-peer protocols to control and operate these bots.

Botnets have multiple nefarious uses: mounting DDoS attacks, stealing user passwords and identities, generating click fraud [Daswani *et al.*, 2007], and sending spam email [Ramachandran and Feamster, 2006]. This anecdotal evidence suggests that a common strategy for monetizing botnets is sending spam email, where spam is defined liberally to include both traditional advertisement email, as well as phishing email, email with viruses, and other unwanted email.

In this chapter, we present new techniques to map botnet membership and other characteristics of botnets using spam traces. Our primary data source is a large trace of spam email from Hotmail, one of the largest Web mail services. Using this trace, we both identify individual bots and analyze botnet membership (which bots belong to the same botnet). The primary indicator we use to guide associate individual machines with membership in a single larger botnet is participation in *spam campaigns*: coordinated mass emailing of spam. The basic assumption is that spam email messages with similar content are often sent at the command of a single controlling entity, because these email messages share a common economic interest. Through these hidden connections among spam email, we identify a set of botnets. Our approach easily extends to other methods for estimating email similarity, e.g., an OCR algorithm optimized for image spam. Detailed techniques are presented in Section 4.4.

Our focus on spam stands in contrast with much previous work studying botnets. Previous studies have proposed monitoring remote compromises related to botnet propagation [Cooke *et al.*, 2005], actively deploying honeypots and intrusion detection systems [Krasser *et al.*, 2005], infiltrating and monitoring IRC channel

communication [Binkley and Singh, 2006; Cooke *et al.*, 2005; Freiling *et al.*, 2005; Rajab *et al.*, 2006], redirecting DNS traffic [Dagon *et al.*, 2006] and using passive analysis of DNS lookup information [Rajab *et al.*, 2007; Ramachandran *et al.*, 2006].

Focusing on spam has benefits. The analysis can be done on email traces from a large email provider. Spam email gives us new insights into trade-offs in botnet design in practice, such as botnet size versus the average number of message sent per bot. As we discuss here measuring spam allows us to determine whether individual bots are part of the common larger botnet. This complements analyses in previous work [Binkley and Singh, 2006; Cooke *et al.*, 2005; Freiling *et al.*, 2005; Rajab *et al.*, 2006; Dagon *et al.*, 2006; Rajab *et al.*, 2007; Ramachandran *et al.*, 2006] that require monitoring IRC channels or sometimes monitoring DNS caches.

Analyzing a large trace of spam email messages presents a number of technical challenges. These challenges include:

1. It is not easy to identify the "same" spam email sent to different inboxes, as each one is often slightly different. Spammers routinely employ tricks to evade detection by changing each email message.

2. The presence of hosts with dynamic IP addresses means that simply counting the number of originating IP addresses is insufficient to estimate the host population (more generally, one cannot easily infer host equality from IP address equality).

3. Because one botnet may be involved in multiple spam campaigns at the same or different times, estimating the number of botnets involved in sending all observed spam email requires estimating whether two spam campaigns originate from the same botnet or not.

Our work answers these challenges. We found hundreds of botnets by examining a subset of the spam email received by Hotmail. The botnets we found range in size from

tens of hosts to more than ten thousand hosts. The number of bots and botnets found in our study are comparable to those identified in previous work [ShadowServer.Org, 2007]. Basic metrics reported in this chapter are consistent with those from previous work [Rajab *et al.*, 2007]. This demonstrates that our approach is able to catch a significant number of active botnets.

The primary contributions of our approach in this case study are:

- We are the first to analyze entire botnets behavior from spam email messages rather than just the behavior of individual bots. We propose and evaluate methods to identify bots and cluster bots into botnets using spam email traces. Our approach can identify botnets regardless of their internal organization and communication. Our approach is not thwarted by encrypted traffic or customized botnet protocols, unlike previous work using IRC trackers [Cooke *et al.*, 2005; Freiling *et al.*, 2005] or DNS lookup [Rajab *et al.*, 2006; Ramachandran *et al.*, 2006; Rajab *et al.*, 2007].

- We report new classes of information about botnets. For example, we report on the relationship between botnets usage and basic properties such as size. We also confirm previous reports on capabilities of botnet controllers and botnet usage patterns.

We believe our measurement results will be useful in several ways. First, knowing the size and membership gives us a better understanding on the threat posed by botnets. Second, the membership and geographic locations are useful information for deployment of countermeasurement infrastructures, such as firewall placement, traffic filtering policies, etc. Third, characterizing behaviors of botnets in monetizing activities may help in fighting against botnets in these businesses, perhaps reduce their profits in sending spam, generating click fraud, and other nefarious activities. Finally,

such information about botnets may also give law enforcement help in combating illegal activities from botnets.

We also believe that the techniques presented here may also be applicable to related domains, such as identifying botnet membership through click fraud (analogous to spam) identified in search engine click logs (analogous to email traces).

The application of machine learning and data mining to security is an area of great debate: there has been significant recent interest in doing this [Maloof, 2006], while there have also been a number of papers describing challenges with this approach [Barreno *et al.*, 2006; Chan and Lippmann, 2006]. We hope that this case study provides a new and interesting example of how data mining techniques can be applied to security problems.

4.2 Related Work

We study botnets by analyzing spam email received by Hotmail Web mail service. Our work thus provides an approach to studying global Internet behavior (botnets, in this case) through a single observation point. Several previous studies [Anderson *et al.*, 2007; Ramachandran and Feamster, 2006] also use spam email collected at a single or small number of points to gain insight into different aspects of the Internet.

Spamscatter [Anderson *et al.*, 2007] studies the scam hosting infrastructure by following the embedded links in spam email and clustering snapshots of the destination Websites. Spamscatter uses image shingling (e.g. comparing whether small portions of images are identical) to determine spam campaigns. Their study produced valuable findings about the geographic distribution, host lifetimes, degree of sharing among hosts, and other aspects of the scam hosting infrastructure. Our work is different in that we cluster email based on content and study the source (sending) infrastructure instead of the destination websites linked to from the spam email.

Ramachandran and Feamster [Ramachandran and Feamster, 2006] collect spam email by creating a sinkhole and study network-level behavior of spam, such as the IP ranges that generate the most spam and the spam-generating lifetimes of hosts. They also provide a case study of the Bobax botnet by intercepting its "command and control" IRC traffic. Both their work and ours study the interaction between spam and botnets. However, their work is more about characteristics of bots in general, and studies network-level characteristics among all email and sender IP addresses (or bots). Only in the case study of one specific botnet, Bobax, is botnet membership monitored, using analysis of IRC traffic.

In contrast, our work identifies the membership of many botnets and compares metrics across them, such as botnet size, number of active hosts in a botnet and spam email sent per bot in a botnet. One of the major results of our work is detailed information about botnet membership, which is not included in the previous work. We also take into consideration IP dynamics when detecting bots and botnets, unlike some previous work. Work on IP dynamics [Xie *et al.*, 2007] shows that a significant portion of IP addresses are reassigned within some time window ranging from a few hours to a couple of days. Our approach is more accurate because the estimation of bot and botnet characteristics is not just simple aggregation among IP addresses.

Our work is also related to literature on botnets in general. Techniques to gather botnets for study fall mainly into two categories [Rajab *et al.*, 2007].

The first category of techniques collects botnets traffic from inside. At least two major approaches have been proposed: IRC channel infiltration and traffic redirection. In the first approach, IRC infiltration, a modified IRC client joins the IRC channel of a botnet and plays the role of a "mole". This mole executes commands from the botnet controller and mimics actual bots, while more importantly monitoring any "command and control" traffic in the channel. The IRC tracking approach has been examined in detail in [Cooke *et al.*, 2005; Freiling *et al.*, 2005]. An algorithm

for anomaly-based botnet IRC traffic detection was proposed in [Binkley and Singh, 2006]. A second approach is traffic redirection, in which DNS traffic related to the command and control server is redirected to a sinkhole for study [Dagon *et al.*, 2006]. The sinkhole intercepts and redirects all connection attempts from bots of a botnet. Data collected using both approaches can be quite accurate. However, both methods must be tailored for each concrete botnet implementation. In particular, infiltration requires significant effort to modify the bot client for each botnet, especially for those botnets using customized protocols instead of IRC command and control channels. As both approaches actively probe botnets traffic, it is possible for the botnet controllers to discover the probing. These approaches known by botnet controllers, so botnet controllers may disable broadcast feature on their IRC channels or suppress bot identity information from IRC traffic.

The second category of techniques track botnets from external traces. Cooke et al. suggested that secondary bot behavior such as propagation and attacks should be used [Cooke *et al.*, 2005] in botnet detection. A set of DNS-based botnet detection approaches [Rajab *et al.*, 2006; Ramachandran *et al.*, 2006] exploits one specific type of external trace, i.e. DNS lookup information. Rajab et al. exploits the fact that most bots issue DNS queries to resolve the IP addresses of their IRC servers and then probe caches of a large number of DNS server to infer the footprint of a particular botnet [Rajab *et al.*, 2006]. In [Ramachandran *et al.*, 2006], DNS blackhole list (DNSBL) lookups are used to detect botnet membership, since attackers frequently query DNS blacklists to find out which of their hosts are blacklisted. Krasaridis et al. [Karasaridis *et al.*, 2007] present a method for detecting botnets by employing scalable non-intrusive algorithms that analyze summary traffic data collected on selected network links (mostly at the transport layer).

Our work falls into the second category, and it exploits another type of external trace from botnets: actual received email spam. Spam email is readily available for

analysis at certain observation points (for example, mail service providers). This data source is interesting because it is relatively easy to collect and is comprehensive in nature. In contrast, DNS probing [Rajab *et al.*, 2006; Ramachandran *et al.*, 2006; Rajab *et al.*, 2007] requires extra queries to DNS servers, and could be limited by the querying rate to DNS servers. Botnet detection from flow data across a large Tier 1 ISP network [Karasaridis *et al.*, 2007] demands extraordinarily data collection efforts. While previous work focuses on traffic generated by botnets, our work is the first to study botnet traces based on economic motivation and monetizing activities. Our work also reports on more metrics than previous work.

4.3 Overview

This section presents an overview of our approach. Section 4.4 presents the techniques in detail.

Our technique takes as input a set of email messages that have been labeled as spam; in the particular data set that is the main focus of Section 4.5, these labels are human-generated. From each spam email message, we extract its sender IP address, sending time and content. We assume spam email messages with the same specific topic are sent from the same controlling entity, and then detect botnet membership in three steps.

1. **Cluster email into spam campaigns.** We compute a set of fingerprints (e.g. 10 in our case) from the content of each spam email. The fingerprint set is a digest of the email content. If the fingerprints of two spam email messages overlap significantly, these email messages have the same or near-duplicate content. A set of email that shares common fingerprints are clustered together. In this way, we cluster email with the same or near-duplicate content

into a *spam campaign*. Intuitively, a spam campaign is a set of email with the same content or almost the same content, or other connections such as linking to the same target URL.

2. **Assess IP dynamics of each C-subnet.** For each class C-subnet, we extract 1) the average time until an IP address gets reassigned; 2) the IP reassignment range. Using these parameters, we propose a way to estimate the probability whether two events $((\mathrm{IP}_1, t_1)$ and $(\mathrm{IP}_2, t_2))$ are initiated from the same machine.

3. **Merge spam campaigns into botnets.** Based on the first two steps, we merge individual spam campaigns together into a set of spam campaigns initiated by the same *botnet* if the sending hosts significantly overlap. For each spam event in a spam campaign (SC_1), we use the previously-calculated IP dynamics to estimate the likelihood that the host sending the spam also participates in another spam campaign (SC_2). Then, if a large number of senders participate in both spam campaign SC_1 and spam campaign SC_2, we merge the two together.

   To estimate the number of machines in a botnet, we further leverage the previously-calculated IP dynamics. A machine may send several spam email messages at different times, with different IP addresses appearing in the corresponding spam events because of IP reassignment. We describe an approach in Section 4.4 to remove the IP reassignment effect and then estimate the actual membership of the botnets.

4.4 Methodology

We first define a set of terms we will use in the following discussion.

- A *spam* email is an unsolicited bulk email, often sent to many people with little or no change in content. In the particular dataset that is the main focus of

Section 4.5, the spam label is human generated.

- A *spam campaign* is a set of email messages with the same or almost the same content, or content that is closely related—for example, linking to the same target URL.
- A *botnet* is a set of machines that collaborate together to run one or more spam campaigns.

In this section, we discuss in detail our approach to extracting botnet membership by analyzing spam email data.

4.4.1 Datasets and Initial Processing

We work on two email datasets collected in different ways[1] from Hotmail.

The first dataset (referred to as "Email Samples (ES)" later) is a uniformly distributed sample of all email received by Hotmail every day. This sampled set is reviewed by a group of volunteer users who are willing to mark whether an email is spam or not based on their personal subjective decisions, and the non-spam email is removed. In this way, the ES dataset gives us a large daily corpus of spam email.

The other dataset (referred to as "Junk Mail Samples (JMS)" later) is collected from Hotmail using a different method. JMS is collected from email in inboxes that is reported as spam (or "junk") by users. To derive our estimates (see Section 4.4.7), we use the approximation that spam labeled email is a uniform subsample of all spam received. A previous study similarly used an email spam sample derived by this technique to estimate global spam behavior [Hulten *et al.*, 2004]. We only know that the JMS dataset contains a fixed percentage of all "junk" reports, so we need to estimate the sample rate of the JMS dataset among all spam email. To do this, we

[1] To perform our study, we collected approximately ten million spam email messages from a large web email service.

compare the size of the JMS dataset and the ES dataset: we will call the ratio of the JMS dataset size to the ES dataset size s and the sample rate of the ES dataset r $(0 < r < 1)$. We also know the fraction f of email spam that Hotmail detects before it reaches the user's inbox. Supposing that the total number of spam email messages received by Hotmail is N, we get the following equation:

$$N * r * s = N * (1 - f) * \text{jms-real-sample-rate}$$

This gives us the sample rate of the JMS dataset among all spam email. The sample rate will be used in Section 4.4.7 to estimate total size of botnets.

Related work [Anderson *et al.*, 2007] indicates that one week is a reasonable duration to analyze spam campaigns, given the fact that spam campaigns change fast over time. To support our analysis, we used the ES and JMS datasets for the nine-day period from May 21, 2007 to May 29, 2007. The amount of spam email identified during the nine-day period was on the order of ten million messages. It is about the same size as that used in [Ramachandran and Feamster, 2006] (collected over 1.5 years) and one order of magnitude larger than that used in [Anderson *et al.*, 2007] (collected during 7 days). The email messages are in a raw format with all headers and all MIME body parts included. We parse each email to get the sender IP addresses from the header. We use a method similar to the one discussed in [Brodsky and Brodsky, 2007] to identify each email's source IP address. Basically, we trust the sender IP reported by Hotmail in the *Received* headers, and if the previous relay IP address (before any server from Hotmail) is on our trust list (e.g. other good mail service), we continue to follow the previous *Received* line, till we reach the first unrecognized IP address in the email header. This IP address is then assumed to be the email source. We also parse the body parts to get both HTML and text from each email. After some initial processing, we extract sending time and content

(HTML/plaintext), along with sender IP address, for each email.

4.4.2 Data Analysis Infrastructure

Data analysis of massive data is much easier nowadays, thanks to tools in statistical learning and new advancement on distributed infrastructures for storage and computation.

Distributed storage and computation infrastructure for massive data analysis have been of huge interests in the systems area in recent years. In 2004, Google first discussed the Google File System (GFS) that is running across thousands of machines in a cluster environment [Ghemawat *et al.*, 2003]. GFS provides a transparent interface that application can view it as a single huge filesystem (e.g. in petabytes) without knowing underlie implementation details. GFS is an append-only filesystem. GFS made it possible to store huge data set and retrieve the data sequentially. In 2005, Google further discussed a distributed computation infrastructure called MapReduce that transparently distributes data processing tasks over thousands of machines in the same cluster [Dean and Ghemawat, 2004]. Similar systems include Dryad [Isard *et al.*, 2007] and a open source project called Hadoop [Hadoop, 2007], etc. We leverage these infrastructures to facilitate data analysis. We are able to analyze security problems efficiently using massive data set, which was not possible in the past.

In this case study, we use a system similar to MapReduce as our computation infrastructure to process the dataset.

4.4.3 Identifying Spam Campaigns

A spam campaign consists of multiple related email messages. The messages in a spam campaign share a set of common features, such as similar content, or links (with or without redirection) to the same target URL. By exploiting this feature, we

can cluster spam email with same or near-duplicate content together as a single spam campaign.

Since many anti-spam filters work by searching for patterns in headers or bodies of email messages, spammers obfuscate the message content such that each email in a spam campaign has slightly different text from the others. One common obfuscating technique is misspelling commonly filtered words or inserting extra characters. HTML-based email offers additional ways to obfuscate similarities in messages, such as inserting comments, including invisible text, using escape sequences to specify characters, and presenting text content in image form, with randomized image elements.

Because of this message content obfuscation, identifying whether two email messages are from the same campaign is not trivial. The algorithm to cluster spam email messages with the same or near-duplicate content must be robust enough to overcome most of the obfuscation. Fortunately, most obfuscation does not significantly change the main content of the email messages after being rendered, because it still needs to be readable and deliver the same information. Thus, we first use ad hoc approaches to pre-clean the content, and then compare parts of the email messages (as suggested by [Broder *et al.*, 1997] in his shingling algorithm). We briefly describe our approach below.

We first apply a sophisticated HTML parser to the HTML portion of the email body. The HTML parser returns the actual content rendered to users, which we will refer to as *real text*. We remove extra spaces, line breaks, and similar extraneous characters from the text. We also extract embedded HTML links from the email. Later we use the HTML links together with cleaned text to measure content similarity.

After this pre-cleaning step, the real text is used to compute a "fingerprint". We compute fingerprints from all possible substrings of length l (we call these substrings *text chunks*) [Manber, 1994]. The fingerprints are "digests" computed as follows [Ra-

bin, 1981]. Denote the text string by $t_1 t_2 \cdots t_n$. The fingerprint for the first text chunk will be:

$$F_1 = (t_1 \cdot p^{l-1} + t_2 \cdot p^{l-2} + \cdots + t_l) \bmod M,$$

where p and M are constants. Thus, the fingerprint for the second text chunk will be:

$$F_2 = (t_2 \cdot p^{l-1} + t_3 \cdot p^{l-2} + \cdots + t_{l+1}) \bmod M = (p \cdot F_1 + t_{l+1} - t_1 \cdot p^{l-1}) \bmod M.$$

Continuing thusly, we compute the fingerprint of each text chunk based on the fingerprint of the previous text chunk, getting $n - l + 1$ fingerprints in total.

Next, we deterministically select m fingerprints out of all $n - l + 1$ fingerprints. In our experiment, we select the m largest fingerprint values from the entire set. We set the text chunk length l to 50 and m to 10 (these values were suggested in an earlier study [Zhou *et al.*, 2003]). If five matching fingerprints are found out of the ten generated for each of two text strings, we regard them as having *connected content*. Selecting the number of matching fingerprints presents a tradeoff between false positives and false negatives. If we use a larger number of matching fingerprints, we are less likely to group two different email messages together (i.e. a false positive). If we use a smaller number of matching fingerprints, we are less likely to miss email messages that are actually similar (i.e. a false negative). In previous work [Zhou *et al.*, 2003] we found that the threshold of five fingerprints provides sufficient tolerance to content modifications in email messages while keeping the false positive rate extremely low.

Now, we consider each email message as a node in a graph, and draw an edge between two nodes if the corresponding two messages are connected in content (e.g. have at least five matching fingerprints) or share a single embedded link. Email service providers sometimes add advertisement tails to messages, which often contain

embedded links. These links appear in non-spam email messages as well as spam email messages. We build a whitelist using this feature to filter out links in these tails by domain or site names. After removing these popular tailer links, we find that even a single shared link is a strong evidence of connection among messages. We then define each connected component in the graph as a *spam campaign*. Using the Union-Find algorithm [Cormen *et al.*, 2001], we can label all connected components on the graph, with each label representing a spam campaign. We can thus generate a list of detected spam campaigns. To assign labels, we associate each spam campaign with the list $\{(\text{IP}_i, t_i)\}$ of *IP events* consisting of the IP address IP_i and sending time t_i (see Section 4.4.1) extracted from each email message in the campaign.

We use only one approach (i.e. text shingling) to group email messages into spam campaigns. However many others exist (and will be developed in the future to adapt to changing characteristics of email campaigns) and may be used together with text shingling as part of the overall approach described in this paper[2].

4.4.4 Skipping Spam from Non-bots

Spam that comes from non-bots is excluded from our study.

- We build a white list of IP addresses, which includes known email service providers, ISP MTA servers, some popular proxies, open relays, etc. If the sender IP address of a message (e.g. the first *Received* header) is on the white list, we exclude that email from further analysis.

- We also remove campaigns whose senders are narrowly located in a small IP range, which could be an IP range controlled by a spammer. Sender IPs that

[2]For example, text shingling misses image spam messages and associated spam campaigns. URL-based detection may also miss spam campaigns that contain different URLs in messages that redirect to a common website. See [Anderson *et al.*, 2007].

show a wide range of IP addresses indicate that the spam campaign was sent through different ISP providers and networks.

- We only include campaigns whose senders are from at least three geographic locations.

Hotmail blocks most spam messages from spammer servers and many open relays using volume-base policies and excludes these messages from user inboxes. Since the JMS dataset includes a sample of spam in inboxes, we rule out most (if not all) spam from non-bots using approaches above. It is very likely that spam campaigns that originate from hundreds or even thousands of geographic locations are operated by botnets. We can also characterize spam coming from smaller numbers of geographic locations, allowing us to conduct further analysis in future work.

4.4.5 Assessing IP Dynamics

In order to obtain accurate metrics on spam campaigns and botnets, we need to know whether two IP events (IP_1, t_1) and (IP_2, t_2) $(t_1 \neq t_2)$ occurring at different times originate from the same machine or not. If all IP addresses were static, then determining the answer to this question would simply be a matter of comparing IP_1 and IP_2 for equality. Unfortunately, a significant number of IP addresses are dynamic.

Many home computer users currently connect to the Internet through dial-up, ADSL, cable or other services that assign them new IP addresses constantly—anywhere from every couple of hours to every couple of days for most common cases. We need to know how dynamic each IP address is, before we can know whether to "merge" it with another IP address in the same spam campaign [Xie *et al.*, 2007].

We begin by assuming that within any particular class C-subnet (256 consecutive IP addresses differ only in the last 8 bits), the IP address reassignment strategy is uniform. We also assume that IP address reassignment is a Poisson process (a

stochastic process used for modeling random events in time that occur to a large extent independently of one another) and measure two IP address reassignment parameters in each class C-subnet: the average address lifetime J_t on a particular host, and the maximum distance J_r between IP addresses assigned to the same host.

The dataset from which J_t and J_r are measured is the log of 7 days' user login and logout events (June 6-12, 2007) from the MSN Messenger instant messaging service. For each login and logout event, we obtain an anonymized username and IP address for that session. We associate login and logout events for the same username to construct a sequence:

$$\text{username}: \begin{array}{l} (\text{IP}_1, [\text{login-time}_1, \text{logout-time}_1]), \\ (\text{IP}_2, [\text{login-time}_2, \text{logout-time}_2]), \\ (\text{IP}_3, [\text{login-time}_3, \text{logout-time}_3]), \\ \ldots \end{array}$$

We assume that each user connects to the MSN Messenger service from a small, fixed set of machines (e.g. an office computer and a home computer), and detect cases where multiple IP addresses are associated with a particular username. We label each such change as an IP address reassignment if the IP addresses are sufficiently "close": we define "close" as within a couple of consecutive class B-subnets (i.e. IP addresses differ only in the last 16 bits); otherwise, we assume that two different machines are involved. We then aggregate our detection among all IP addresses in the same class C-subnet and remove anomalous events. We then calculate, based on the Poisson process assumption, J_t and J_r for each individual class C-subnet.

Thus, given two IPs at two different times, (IP_1, t_1) and (IP_2, t_2), if either IP_1 or IP_2 is out of the distance range (J_r) of another, we regard these two events as from two different machines. If both IP_1 and IP_2 are within the distance range (J_r) of each

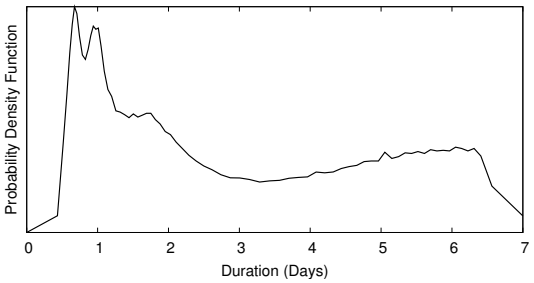

Figure 4.1: Probability density function of IP reassign duration

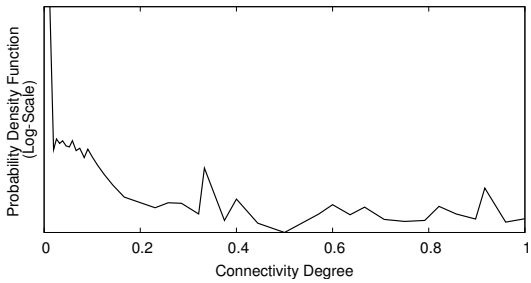

Figure 4.2: Probability density function of the campaign merge weight

other, we make the computation below.

$$\mathbb{P}[\text{IP}_1 = \text{IP}_2| \text{ actually the same machine}] = \frac{J_r - 1}{J_r} * \exp\left(\frac{-(t_2 - t_1)}{J_t}\right) + 1/J_r = w(t_1, t_2).$$

This is the probability that a machine has kept the same IP address after an interval of duration $t_2 - t_1$.

$$\mathbb{P}[\text{IP}_1 \neq \text{IP}_2| \text{ actually the same machine}] = \frac{J_r - 1}{J_r} * \left[1 - \exp\left(\frac{-(t_2 - t_1)}{J_t}\right)\right] = 1 - w(t_1, t_2).$$

This is the probability that a machine changes its IP address – that is, that an IP reassignment happens – during an interval of duration $t_2 - t_1$.

We define *IP reassignment time* as the mean period of time that a single machine holds an IP address. Figure 4.1 shows a probability density function of IP reassignment time among all class C-subnets (about 25% of class C-subnets never see IP reassignment in the 7 day log). According to our data, a large portion of IP addresses get reassigned almost every day.

4.4.6 Identifying Botnets

Each spam campaign is represented as a sequence of events (IP, t), where each event is a spam email that belongs to the spam campaign. The question is, given two spam campaigns SC_1 and SC_2, how do we know whether they share the same controller (i.e. they are part of the same botnet)? We put two spam campaigns in the same botnet if their spam events are significantly connected.

Given a event (IP_1, t_1) from spam campaign SC_1 and a event (IP_2, t_2) from spam campaign SC_2, we assign a connection weight between them. The connection weight is the probability that these two events would be seen if they were actually from the same machine. Section 4.4.5 defines $w(t_1, t_2)$ if two IP addresses are equal, or $1 - w(t_1, t_2)$ if two IP addresses are not equal but within distance range of each other, or 0 otherwise. For all events in a spam campaign SC_1, we use

$$W = \frac{\sum_i \max_j [w(t_i, t_j) \text{ or } (1 - w(t_i, t_j)) \text{ or } 0]}{|\text{SC}_1|}$$

to measure the fraction of events in SC_1 that are connected to some events in SC_2, where i and j represents IP events in SC_1 and SC_2. W is called the *connectivity degree*.

We use the connectivity degree W to decide whether we should associate different spam campaigns with a single botnet. Here we explain why we define W in the format above.

- W intuitively attempts to represent the fraction of events in a spam campaign that are related to another spam campaign. Therefore, we expect W ranges from 0 to 1. If W goes to 1, it means a significant portion of the events in SC_1 are connected to events in SC_2, and thus, we should merge SC_1 into SC_2. The definition above satisfies this requirement.

- At first, we may expect the definition of W to be symmetric. For example, one possible symmetric definition would be as follows:

$$W = \frac{\sum_i \sum_j [w(t_i, t_j) \text{ or} (1 - w(t_i, t_j)) \text{ or } 0]}{|\text{SC}_1||\text{SC}_2|}.$$

  However, an asymmetric definition of W is more proper in our scenario, because we want to compare the relationship between a pair of sets. The metric should measure the portion of set SC_1 that is related to set SC_2, which is not symmetric by definition. For example, when SC_1 is a subset of SC_2, we want to merge all events in SC_1 to SC_2. We require W as 1 in this scenario and the symmetric definition does not satisfy this requirement.

- We compute W values from each spam campaign to all other spam campaigns in the JMS dataset. We expect most W values are small, which indicate one spam campaign is not connected to another spam campaign; while a small portion of W values are relatively large, which indicate one spam campaign is connected to another spam campaign. There should be few W values in the middle. We expect to see a bimodal pattern in the probability density function of W values. Figure 4.2 shows the probability density function of W values computed from the JMS dataset. The curve in Figure 4.2 matches our expectation and indicates that a W value in the range between 0.05 and 0.35 is a reasonable threshold to merge spam campaigns.

- We expect our detection is not sensitive to the threshold of W. We select 0.2 as a reasonable threshold to decide whether spam campaigns should be merged. We use the botnet detection results of this value as a baseline for comparison. In our experiments, we compare baseline results with results of thresholds from 0.05 to 0.35. We find that the change of threshold has very little effects to the

botnet detection results. Because the detection is not sensitive to the threshold, it gives us more confidence in the validity of the clustering.

- We also examined alternative, simpler definitions of W, by counting pairs of matching IP addresses. However, IP address matching is insufficient to merge all related spam campaigns together. We have found spam campaigns from a single botnet but are not merged by simple IP address matching.

The connectivity degree W is also related to the way that botnet controllers use their botnets. If a botnet controller always use all its bots to run each spam campaign, we will observe that each spam campaign has $W = 1$ to other spam campaigns from this botnet. However, as we will show in Section 4.5.2 botnet controllers use only a subset of available bots each time.

If a spammer uses two individual botnets for the same spam campaign, there is a chance that our strategy will incorrectly merge two individual botnets into one. We define a botnet as a set of machines that collaborate together. A manual examination of the detected botnets shows that the definition W and the merging strategy works well: most pairs of IP addresses in a single botnet participate together more than one spam campaigns at different time. This indicates our selection of W works well under our definition of botnets: if a pair of IP addresses were actually from two botnets, we should not see their collaboration in multiple campaigns over time.

Spuriously introduced IP addresses in spam campaigns may add noise to W. In our experience, the detection result is not sensitive to the threshold of W: the value W can tolerate up to 15% spurious events in a spam campaign according to the parameter selection above. Each spam campaign contains hundreds of thousands of events. It is hard to introduce a large number of spurious events without being discovered. In practice, a few number of spurious events does not affect the results.

4.4.7 Estimating Botnet Size

After identifying botnets (see Section 4.4.6), each botnet contains a sequence of events (IP, t) that correspond to all spam sent by this botnet. We want to identify distinct machines that generate these events. In Section 4.4.5, we have already defined the probability that two events are from the same machine. We use this definition to examine events in a botnet: when an event (IP_2, t_2) is from the same machine of a previous event (IP_1, t_1), IP_2 is a reoccurrence of IP_1. So, we can estimate the probability that an IP address is a reoccurrence of any previous IP address:

$$c = 1 - \prod_i \mathbb{P}[\text{IP is not a reoccurrence of IP}_i],$$

where i ranges over all events that happened before this IP event. The value of c equals 1 if the IP address is a reoccurrence, 0 if the IP address is not a reoccurrence. In this way, we can count the number of distinct machines appeared in the downsampled dataset (JMS).

Furthermore, we want to estimate the total size of botnets from the downsampled dataset (JMS). We assume bots in the same botnet behave similarly — each bot sends approximately the same number of spam messages. As we explain in Section 4.4.1, we assume the dataset is uniformly downsampled and estimate overall sample rate of the JMS dataset as 0.001.

We define the following quantities:

- r: sample ratio of the dataset among all email received by this large mail service
- N: number of spam email messages observed
- N_1: number of bots observed with only one spam email in the dataset

We want to measure botnets size and number of spam email messages sent per bot:

- s: the mean number of spam email messages sent per bot
- b: number of bots (i.e. botnet size)

For s spam messages sent from a bot, with a sample rate r, the probability that exactly one spam message appears in our dataset is

$$r * (1-r)^{s-1} * \binom{s}{1}.$$

If the number of bots is b, the expected number of bots observed with only one spam email message in the dataset is

$$b * \left[r * (1-r)^{s-1} * \binom{s}{1} \right],$$

which is N_1 according to our definition above. We also know that the expected number of spam email messages from a botnet is $N/r = s * b$. Note that $N = s * b * r$, so,

$$N_1 = b * \left[r * (1-r)^{s-1} * \binom{s}{1} \right] = s * b * r * (1-r)^{s-1} = N * (1-r)^{s-1}.$$

The number of spam email messages sent per bot (s) and botnet size (b):

$$s = \frac{\log(N_1/N)}{\log(1-r)} + 1,\ b = \frac{N}{rs}$$

The assumption that each bot sends approximately equal number of spam messages works well for many botnets. However, it is an unverified assumption and the conclusions that follow are conditional upon the validity of this unchecked assumption. Verifying this assumption poses challenges, but with capture of a significant fraction of email traffic, an observer could estimate the number of email messages sent by a single machine. Currently, we expect that the number of messages sent per bot would

not vary significantly and the assumption would work well for most botnets. This is because each botnet controller uses specific vulnerabilities to compromise machines — it is likely that bots in a botnet are machines with similar processing power and network bandwidth.

4.5 Metrics and Findings

In this section, we study characteristics of botnets and their behavior in sending spam. These metrics are measured on spam campaigns and botnets detected as described in Section 4.4.

4.5.1 Spam Campaign Duration

Instead of compromising machines and creating botnets themselves, spammers often rent currently existing botnets from the market. How long is a botnet being rented to run a single spam campaign? To answer this question, we must know how long spam campaigns last, i.e. the *spam campaign duration*.

We define the *spam campaign duration* as the time between the first email message and the last email message seen in a given spam campaign in our datasets (JMS & ES). In the calculation, we remove 7% of the spam email messages in the JMS dataset because an insufficient amount of similar spam email messages (where similarity is calculated by shingling, see Section 4.4.3) appeared for us to confidently group it into a single spam campaign. We performed a similar calculation on the ES dataset, and found results consistent with the JMS dataset – we therefore focus only on the JMS dataset for the rest of the discussion.

In this section, we look at all spam campaigns that start on a specific day, and study the duration of these campaigns in our datasets. The cumulative distribution

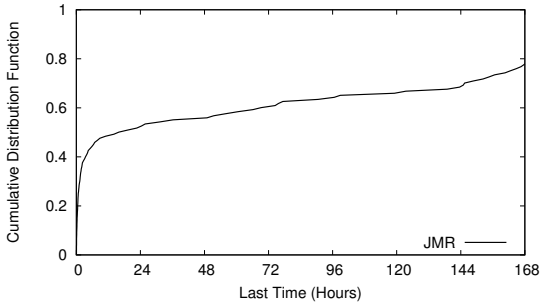

Figure 4.3: Cumulative distribution function of spam campaign duration

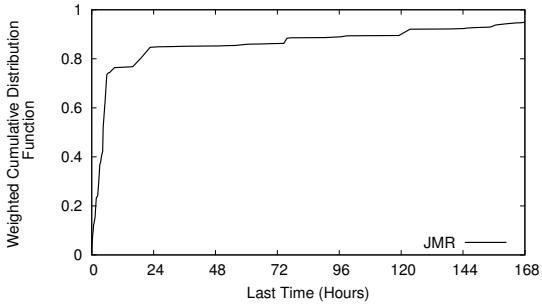

Figure 4.4: Cumulative distribution function of spam campaign duration weighted by email volume

function of spam campaign duration in the JMS dataset is shown in Figure 4.3. The dataset contains email from a nine-day period. For those spam campaigns appearing on the first day, we do not have enough information to know their actual start time from the dataset itself. Because of this, we use only spam campaigns that start to appear on the second day (that is, no appearance on the first day) to calculate the distribution. We find that about 20% of spam campaigns in the JMS dataset persist through the last day of our observation window and they might continue beyond the time window of our dataset. In these graphs, we draw the cumulative distribution function line in the time window from zero hour to 168 hours (seven days).

Figure 4.4 shows the cumulative distribution function of each spam campaign, weighted by email volume. This figure demonstrates the distribution of spam campaign duration in terms of spam email volumes. By comparing the cumulative distribution function and the weighted cumulative distribution function, we find that short-lived spam campaigns have large volume and are often found to send a large amount of email during a short time window.

While over half of spam campaigns end within one day, over 20% of spam campaigns in the JMS dataset that continue through our entire observation window (i.e. nine days). These long-lived spam campaigns have larger effects from the point of

view of the email receiver (as we will discuss in section 4.5.3).

4.5.2 Botnets Size

The capability of a botnet controller is related to number of bots in control. The capability of botnet controllers and level of activity of botnets are two important metrics for understanding botnets. To measure the capability, we need to estimate the total size of each botnet based on our nine days of observation. To measure the level of activity, we estimate the active working set of each botnet in a one hour window. As botnet population is dynamic over time, we use "botnet size" to refer to the estimated number of bots actually used for activities during the time window. This size is estimated as explained in Section 4.4.7. Infected machines are often not cleaned for several weeks. During the period of infection, machines have activities at least every few days. Thus, bots actually used during an observation window of nine days give a good approximation of the number of machines controlled by a botnet controller.

We detected 294 botnets in the JMS dataset and the following measurements are based on these 294 botnets. We estimate the total size of botnets as described in Section 4.4.7. The estimated total sizes of the botnets give us some idea of an upper bound on the capabilities of spammers or botnet controllers – they likely have only compromised this many machines total. We compute 90% confidence intervals of botnet sizes and find that the range of the confidence intervals is less than 20% of the estimated botnet sizes.

Figure 4.5 shows the cumulative distribution function of estimated botnet size[3]. In our dataset, the estimated total sizes of botnets ranges from a couple of machines to more than ten thousand machines; about 50% of botnets contain over 1000 bots

[3]This is the estimation of the number of bots actually used, not just those seen in our dataset.

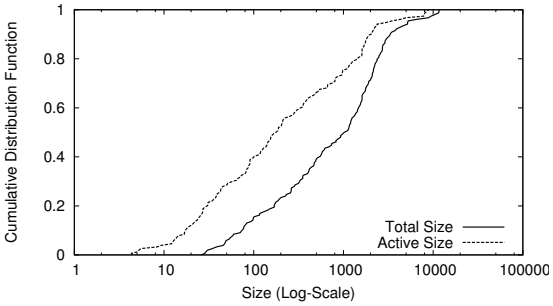

Figure 4.5: Cumulative distribution function of botnet size

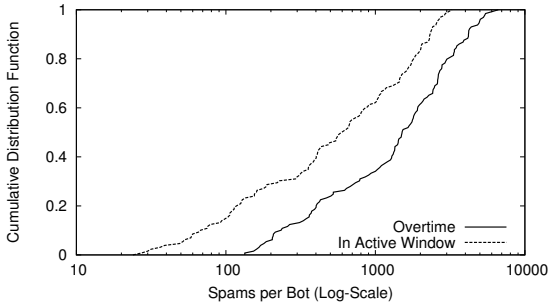

Figure 4.6: Cumulative distribution function of spam email messages sent per bot

each, which is consistent with a similar metric in [Rajab *et al.*, 2007]. The number of spam email messages sent per bot ranges from tens to a couple thousands during the nine-day observation window (Figure 4.6). Some botnet controllers are conservative in the number of spam email messages sent per bot, with about ten messages from each bot.

4.5.2.1 "Average Active Size per Botnet" and "Average Number of Spam Email Messages Sent per Active Bot"

In a time window t (one hour in our experiments):

- "Active size" of a botnet is defined as the number of machines/IPs used for sending spam email by this botnet during this time window t.
- "Spam sent per active bot" in a botnet is defined as the number of spam email messages sent from each bot in a botnet during this time window t.

We now use these two metrics to measure the activity of a botnet.

In the experiment, we study events of a botnet in each time window t during the nine-day duration. Since we limit t to one hour, we can reasonably assume that IP reassignment does not happen. To measure the active size and number of spam email sent per active bot during all time windows (one hour each), we calculate

characteristics in each time window and then average results over all time windows during the nine-day period.

IP blacklisting or volume-based filtering per IP are candidate techniques used for anti-spam in practice. Email service providers constantly update the blacklist, while spammers work hard to avoid getting on the list. In such a situation, the active size of a botnet and the number of spam email messages sent per active bot has a strong impact on the efficiency and effectiveness of these two techniques in filtering spam sent by botnets. Spammers generally use two methods to evade IP based filtering: 1) they send fewer spam messages per bot (which looks like legitimate use); 2) they use a small portion of machines at one time and round-robin among all machines in their control.

Figure 4.7 shows the relationship between the average active size of a botnet and the number of spam email messages sent per active bot. In this figure, we see that large-size botnets tend to send less spam per bot, small-size botnets tend to send more spam per bot, while mid-size botnets behave both ways. This suggests that spam controllers may have clear plans about the number of spam messages to be sent and then stop after these goals are met. Alternatively, the number of email addresses that spammers possess may limit the total number of spam messages sent from their botnets.

On the other hand, Figure 4.8 shows that there is no significant relationship between active botnet duration and the number of spam messages sent per bot. Taken together with Figure 4.7, we conclude that botnet size is the primary factor that determines the number of spam messages sent per bot.

4.5.2.2 Average Relative Active Size

The *relative active size* is defined as the ratio of active size to estimated total size of a botnet. The relative active size in each time window (one hour) is calculated

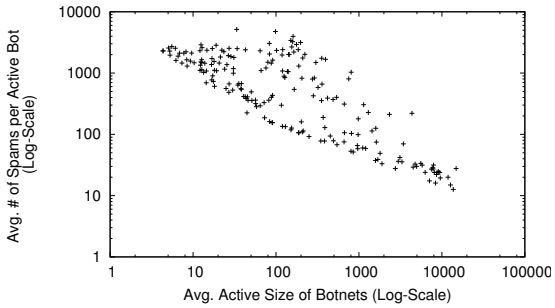

Figure 4.7: Average active size of botnets vs. average number of spam email messages sent per active bot

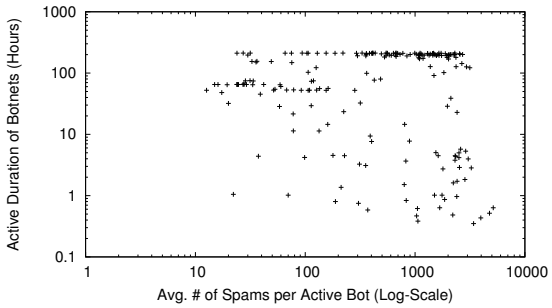

Figure 4.8: Average number of spam email messages per active bot vs. active duration of botnets

and then averaged over nine days. This metric helps us to understand the spamming behavior of botnets and discover different ways spammers use botnets. The average relative active size ranges in (0, 1]. The value of 1 means the botnet uses all machines it controls; while 0 means a botnet uses none of its machines. The average relative active size indicates whether botnets controllers use all machines they have, or use a small fraction of machines and round robin among these machines.

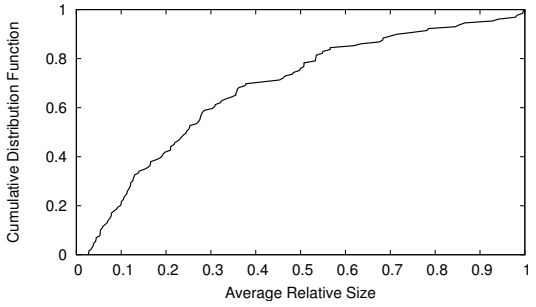

Figure 4.9: Cumulative distribution function of relative active size of botnets

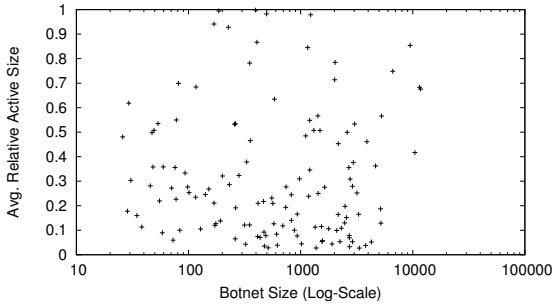

Figure 4.10: Botnet size vs. relative active size

Figure 4.9 is the cumulative distribution function of relative active size of botnets. About 80% of botnets use less than half of bots at a time in their network. Figure 4.10 shows that the relative active size and the total size are not related. That is, in general, a botnet controller might use any portion of bots in his or her control regardless of

the total number controlled.

4.5.3 Per-day Aspect: Life Span of Botnets and Spam Campaigns

If we look at all spam email received in a day by an email server (or an end user), how much spam is from long-lived botnets or spam campaigns? In other word, is it worthwhile to monitor these long-lived botnets? If a new botnet is being rented or borrowed every day for a new spam campaign, monitoring botnets might not be helpful to anti-spam filters. On the other hand, if some botnets are devoted to the spamming business, identifying these botnets is more promising.

We study the duration of botnets and spam campaigns on a per-day basis. We look at spam email received on a particular day, identify the botnets or spam campaigns these spam messages belong to, and compute the distribution of these botnets and spam campaigns with activity on that particular day. In contrast, in Section 4.5.1 we studied the spam campaign duration among all spam campaigns start on a specific day.

In our experiment, we study botnets with activity on the last day of our nine-day observation window, and then look backward to their first activity. We know that each botnet is at least active for x ($1 \leq x \leq 9$) days. Figure 4.11 shows that about 60% of spam received from botnets each day are sent from long-lived botnets. This is a good indication that monitoring botnet behavior, membership, and other properties using the approaches proposed in this paper can help to reduce significantly the amount of spam received on a daily basis.

4.5.4 Geographic Distribution of Botnets

The geographic distribution of botnets is an interesting metric about the ability of botnet controllers in compromising and taking over machines. Figure 4.12 shows

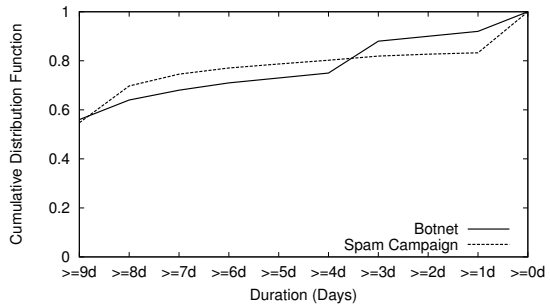

Figure 4.11: Cumulative distribution function of botnets and spam campaign duration from a per-day-activity aspect

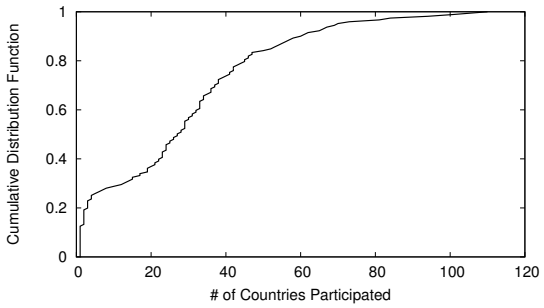

Figure 4.12: Number of countries in botnets

that about half of botnets detected from the JMS dataset control machines in over 30 countries. Some botnets control machines in over 100 countries. Others [ShadowServer.Org, 2007; Wikipedia, 2008b] observe that a botnet typically sends spam messages with the same topic from all over the world, especially from those IP ranges assigned to dial-up, ADSL or cable services. The wide geographic distribution in our results is consistent with their observations.

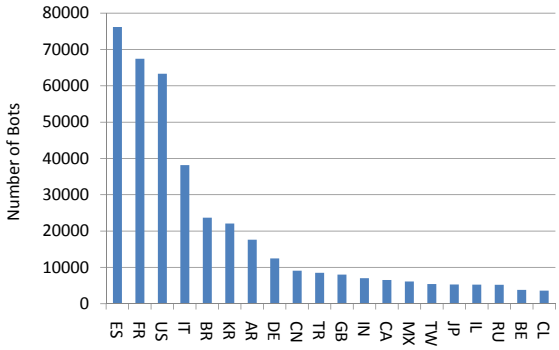

Figure 4.13: Top 20 countries with the largest number of bots

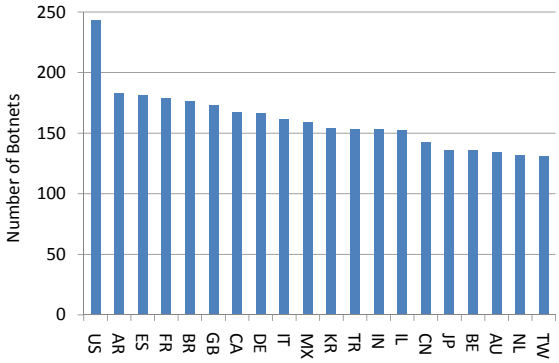

Figure 4.14: Top 20 countries with the largest number of botnets

Using the estimation method proposed in Section 4.4.7, the total number of bots involved in sending spam email all over the world during the nine-day observation period of the JMS dataset is about 460,000 machines. The top twenty countries with

the largest number of bots are shown in Figure 4.13. Figure 4.14 shows the top twenty countries with the largest number of botnets detected from the JMS dataset. It is surprising at first that Spain, France, the United States and Italy have the most bots, since many people report seeing large amounts of spam originating from China, South Korea, and other east Asian countries. As we discussed, our study excludes non-bot spam, e.g. spam sent from spammer servers and open relays. Thus, a large number of bots in a country does not necessarily suggest a large number of spam messages coming from that same country.

4.6 Open Problems

Our work leaves several open problems.

First, we have not provided a ground truth to validate our detection results. Given an IP address, it is hard to verify whether it belongs to a botnet unless we have physical access to the machine that owns the IP address. Our work use intuitive rules to decide whether a set of events are really from botnets: IP addresses of these events must be geographically distributed all over the Internet, the fraction of dynamic IP addresses must be large compared to static IP addresses, etc. In the future, it would be interesting to validate botnets detected from spam email by cross-referencing with results inferred from other techniques such as IRC infiltrating. Comparing with other detection results will also let us know the portion of botnets that do not spam at all and are missed from our approach.

Second, our techniques could strengthen spam filtering. For example, we assign different volume thresholds to senders belong to different botnets given their previous behavior. We may also check the existence of same botnets in query log or ad click log.

Third would be to extend our results to image spam. Certain techniques such as

image shingling are promising for spam clustering.

Finally, it is interesting to further study possible countermeasurements available to botnet controllers to avoid being detected by our approach. Botnet controllers may add even more random variances that make spam email messages different enough from being grouped into spam campaigns. They may also divide a big botnet into multiple smaller ones and use sophisticated strategy for renting. They may be able to introduce errors into our detection. Since this is the first study of its kind, it unlikely that anyone would use countermeasurements against it. It is unknown at this stage how different countermeasurements would affect our approach.

4.7 Summary

Our work opens new directions in understanding botnet activity. By directly tracing the actual operation of bots using one of their primary revenue sources (spam email), we get a picture of bot activity; one that confirms and deepens the understanding suggested by previous work. By identifying common characteristics of user labeled spam email, we associated email with botnets. This allows us to make estimates about the size of a botnet, behavioral characteristics (such as the amount of spam sent per bot), and the geographical distribution of botnets.

This work allows us to collect new classes of information about botnets: as Figure 4.7 shows, we now have much more detailed estimates of the relationship between the active size of botnets and the average number of spam e-mail messages sent by each bot. In particular, we observe that for many botnets there appears to be an inverse relationship between the size of the botnet and the number of spam messages sent per bot.

We are also able to show that the relationship between the geographical distribution of botnets and bots differs in significant ways (Figures 4.13 and 4.14). In par-

ticular, we see that a small number of countries account for most of the bots: Spain, France, the United States and Italy having by far the largest number of bots. In contrast, we see the distribution of botnets is much more uniform. Perhaps, this suggests that most botnets are relatively widely distributed across geographical boundaries, so even countries with a relatively low number of bots still have representation among major botnets.

Our results have wide applicability. First, these results suggest new techniques for email systems designed to catch bot-originated spam. But more than that, they give a forensic insight into the size, geographic locations, amount of gross activities, and amount of activities per bot of specific botnets. This information can be collected dynamically (by analyzing email spam as it arrives) or after the fact – allowing law enforcement to analyze not only current botnets but botnets from previous points in time. Our approach is particularly robust – it does not require creation of new logging mechanisms or depend on continual monitoring of DNS caches. Our approach appears more robust to countermeasures against previous IRC-based botnet monitoring techniques. A botnet can shield messages in IRC exchanges, but as long as a botnet sends out spam messages, it can be analyzed by our approach.

This work is part of a larger effort to develop methods for inferring information from statistical analysis of side-channels (see for example [Zhuang *et al.*, 2005]). Our work leads to some specific opportunities for future research: we have an opportunity to analyze and monitor botnets identified through this work to trace other activities, besides the transmission of spam e-mail messages. This type of monitoring could lead to valuable "early warning" indicators of new classes of malicious undertakings. For example, we could analyze other logged activity (such as search logs, ad click logs, online message logs, firewall logs, exploit logs, etc.) This, in turn, could shed light on the economic motivation of those controlling botnets, and detect links among their actions. Finally, analysis of attacks made by botnets could suggest new classes of

defenses – in particular, they could suggest new methods for detecting a broad class of spam e-mail messages.

Chapter 5

Conclusion

This thesis examined extracting security-related information from large, noisy datasets — the security inference problem. As people develop and deploy more software applications and smart, networked devices, service providers as well as eavesdroppers can collect a growing wealth of digital information, including text logs, email, sound, and even video. On the one hand, this wealth of information increases the risk of leaking important information through unexpected side channels. On the other hand, it also creates opportunities for researchers to better understand many security-related problems. This work presented two case studies that approach the problem from both attacker and defender perspectives. We focused on techniques that extract features reliably from noisy data and then group related samples by correlating similar features.

In both case studies, we followed a five-step framework for security inference. This framework, depicted graphically in Figure 5.1, uses a systematic procedure to group connected samples:

1. Define what constitutes a connection between samples. This generally requires identifying hidden connections within a dataset. These hidden connections are

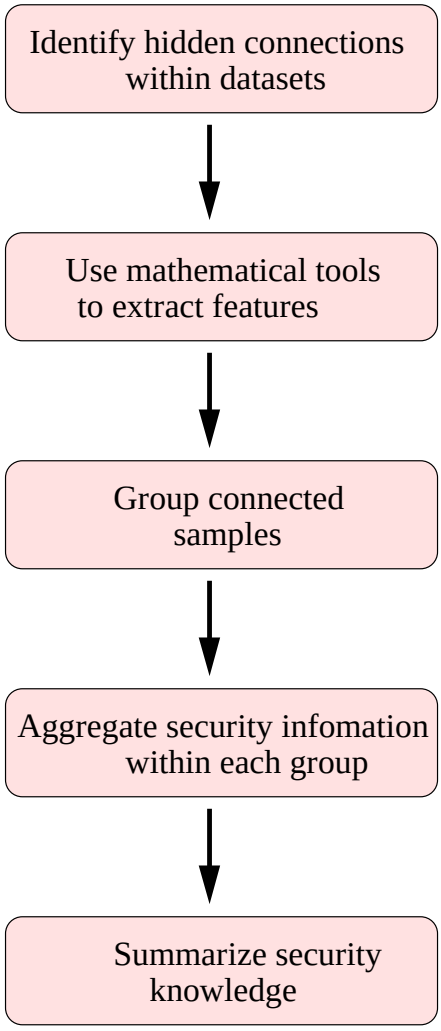

Figure 5.1: A general framework for security inference

used as criteria for grouping samples.

2. Extract features from the data. Several mathematical tools are available to extract features that support finding connections.

3. Group the samples. This step uses statistical analysis and machine learning algorithms based on the extracted features.

4. Aggregate security information within each group. The security information might be simple statistics about the samples in each group or a more complex mathematical model.

5. Summarize security knowledge in a usable form.

This general framework applies to both case studies in this thesis as well as other problems in security inference.

Table 5.1 shows how each case study fits into to the general framework:

- The first case study approaches the security inference problem from an attacker's perspective. The data in this case are sound emissions from typing on a computer keyboard. We identify two hidden connections in the data: 1) keystrokes of the same key sound similar while keystrokes of different keys sound different; 2) typed text is often not random, for example, English speakers mostly type English text. We capture these two observations with mathematic tools. For the first observation, we use frequency features (such as FFT or cepstrum features) during time window around each keystroke in the signal. For the second observation, we use hidden Markov models (HMMs) that represent conditional probability distributions between consecutive characters or words. Unsupervised clustering algorithms on the observations can group keystrokes into a number of acoustic classes based on their sound (represented by frequency features).

	Keyboard Acoustic Emanations	Botnet Detection
Data trace	Sound of typing on computer keyboards	Spam email messages
Identify hidden connections within dataset	1) Different keystrokes sound different; 2) Non-randomness (language constraints) in typed text.	1) Spam email messages in the same campaign share similar content; 2) Bots in a botnet participate in the same spam campaigns repeatedly.
Use mathematical tools to extract features	1) FFT/Cepstrum features; 2) Hidden Markov Models at character level and word level.	Rabin fingerprints
Group connected samples	Sound samples of keystrokes from the same key (Using K-Means, EM and Viterbi algorithm)	Email messages with similar content and spam campaigns sharing senders (Using union-find algorithm)
Aggregate security information within each group	Acoustic information about keystrokes in the same group	Sender identities of email messages in the same group
Summarize security knowledge	Language-independent keystroke classifier	Botnets membership and other metrics

Table 5.1: Applying the general frame to two case studies

Given a recording of sufficient length, HMMs can use language constraints to establish a most likely mapping between these acoustic classes and actual typed characters. The most likely sequence of characters generated from this mapping provides labels for the keystrokes. Using these labels, we group keystrokes from the same key together and train a supervised keystroke classifier. In this way, taking as input a 10-minute sound recording of typed English text, we recover the typed text in this recording as well as create a language-independent classifier. The usable summary of security knowledge in this case is a keystroke classifier, bootstrapped from a sound recording of typed English text, that can recognize even random text such as passwords.

- The second case study approaches the problem from a defender's perspective, discovering security knowledge that can help improve security. The primary data source is a large trace of spam email from Hotmail. Again, two observations reveal connections. First, spam email messages originating from the same controlling entity usually have similar content because these email messages share a common economic interest, so it is likely that a single entity controls the machines sending these messages. Second, bots in the same botnet often participate together in many spam campaigns. We use the Rabin fingerprint algorithm as a mathematical tool to represent content similarity, and group email messages into campaigns. We use the union-find algorithm as a mathematical tool to group spam campaigns together and connect machines to botnets. The security knowledge in this case includes botnet membership and other related metrics.

5.1 Experience with Security Inference

We have presented two case studies that use data mining to extract security information. While the keyboard acoustic analysis case study analyzes data traces of legitimate users, the botnet detection project focuses on data traces of attackers. Experience with these two sides of the problem leads to several important insights:

- Domain knowledge is key to identifying hidden connections among data samples. Successful security inference requires extracting clearly defined security information (the *hidden state*) and knowledge of how it appears in collected data (the link from *hidden state* to *observations*). In the first case study, the hidden states are the actual keys pressed. Domain knowledge informs the assumption that observations of the same key sound similar and different keys sound different. In the botnet study, the hidden state of a computer is its botnet membership. Domain knowledge suggests the assumption that botnet owners send out massive numbers of spam email messages (the observations) with similar content.

- It is useful to review the internal operations of a system and identify all exposed information. For example, normal use of a desktop computer exposes network packets (contents, sending/receiving speed, timing pattern, etc.), optical emanations from the screen, sound from typing on the keyboard, wireless radio if a wireless keyboard or mouse is used, noise from the CPU and machine fans, electricity usage, and other signals. It is useful to consider possible variations in data collected from side channels and how they might correlate with other information. This analysis can highlight features of data samples that are connected and provide a source of side channel information.

- Choosing appropriate tools from the vast repertoire of statistical algorithms and

methods available is a key step in solving the problem. We employ the hidden Markov model in the keyboard study and the "shingling" method for identifying similar texts in the botnet study. For example, after extracting features from data, unsupervised learning methods such as K-means and clustering algorithms are useful to group samples with similar features. The K-means algorithm is useful here because we know the approximate number of clusters (i.e. the k parameter in the algorithm). When we get the labeled samples from bootstrapping recordings, classifiers such as neural networks and Gaussian mixtures can train a classification model, and then give any new sample a label based on the trained model.

Feature extraction is generally a case-by-case issue. A good feature extraction algorithm can significantly improve the learning results. For example, after replacing FFT features with cepstrum features in the keyboard emanation case study, the recovery rate improves by about 10%.

It can also be important to make good modeling choices. For example, when using naive hidden Markov model with Gaussian mixtures in the first case study, the training process encounters the overfitting problem. By using a clustering step first, we reduce the number of unknown parameters in the HMMs and can then successfully estimate parameters using the EM algorithm.

- Flexible and scalable computing infrastructures enable the processing of giant datasets. The ability to extract knowledge from giant datasets is often limited by processing power. This is particularly true in the botnet study, where we analyzed over 10TB of log data. We conjecture that access to a large cluster of computers is critical for many security inference problems.

5.2 Open Problems

This work is a first step in the direction of automatically inferring security knowledge from large, noisy datasets. Challenges ahead arise from rapidly emerging new types of data, increasing quantities of data, and complicated connections across different datasets. Here we suggest several open problems in this area that deserve future study.

1. In the domain of keyboard acoustic emanations, typing could leak other types of information besides just the characters. For example, each user may have different typing patterns and it may be possible to train a classifier that can recognize the identity of the typist in a recording [Monrose and Rubin, 2000]. The information that would be useful for recognizing typist identity includes time between consecutive keys, average typing speed, energy level of typing on each key, etc. It would be interesting to find out: 1) what is the minimum information required to distinguish one user from another; 2) whether the minimum information requirement increases as the total number of users in the database increases; and 3) whether it is easier to verify a user's identity (a yes/no question) instead of classifying a user's identity (choosing among many users).

2. Remote audio (sometimes with video) services such as Skype and Live Messenger are increasing in popularity. As a side effect, remote audio provides a way to collect the sound of typing remotely. If an attacker has both an audio channel and a text channel open with a remote victim at the same time, this setup would provide text-labeled sound samples. For example, an attacker could use a game that employs both voice and text channels simultaneously.

3. Measuring household power consumption reveals information about people liv-

ing inside. Electrical devices have different patterns of power consumption when turned on, running, or turned off. A device that can sample the power consumption at a sufficiently frequent basis can capture this information enabling inference about activities.

4. Social networking websites make new types of data available on the Internet. Public user profiles in many social networks combined with other publicly available data over the Internet (such as news, homepages, forum posts, etc.) can potentially reveal a large amount of information about a person's personal life. Understanding the threats to privacy posed by large-scale correlation of personal information is the first step towards developing strong protections.

5. Query logs and advertisement click logs are promising sources of information about bot behavior. One important question is whether we can distinguish machine (or bot) behavior from human behavior in query logs. As most search engine service providers use user behavior to improve search quality, Web spammers and search engine optimization (SEO) providers often use machines to issue queries, misleading the search engine about user behavior in an attempt to affect ranking strategies of search engines. It is known [Daswani *et al.*, 2007] that botnets are also used to generate fraudulent advertisement clicks. Our success in detecting and characterizing botnets from spam email traces gives hope that similar success may be possible in the domain of advertisement click logs.

6. Cheating players and computers posing as human players (robots) are common in massively multiplayer online role-playing games (MMORPGs) and real-time strategy games. Cheating players and robots behave differently than other users. For example, they may pretend to have high packet loss to move more quickly. Robots respond much faster than human players. We may be able to

detect them by extracting features and building a classifier. Supervised learning and online learning algorithms might be useful here.

7. Our techniques can help detect blog comment spam. Spammers use software to post comments containing advertisements on public blogs. Spammers use comments to attract traffic to certain websites, or to create link farms to boost search engine rank. The content of spam comments is often similar or identical, and URLs in these comments often link to each other. Statistical learning algorithms could use such connections to identify blog comment spam.

8. Similarly, our techniques can help identify web spam. Classifying based on sets of features may identify these spam websites. It is also likely that hosts of websites advertised in spam email messages overlap with web spam hosts websites. Cross-checking datasets from these two services could yield interesting results.

9. Security inference could be applied to analyzing system log data of Internet servers. For example, log data may include messages related to persistent or temporary failures, abnormal volume of particular operations, workload pattern during a day, etc. Can data mining find connections and infer new conclusions from the information inside system logs? We believe it is possible to process these data, discover hidden knowledge inside, and infer relevant information not only for protecting security and privacy, but also for other purposes such as distributed monitoring, system management, bug finding, etc.

Ultimately using security inference to understand massive, noisy datasets enables system designers to find new security vulnerabilities and more effectively track attacker identities and activities.

Bibliography

[Anderson *et al.*, 2007] David S. Anderson, Chris Fleizach, Stefan Savage, and Geoffrey M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *Proceedings of the USENIX Security Symposium*, pages 135–148, 2007.

[Asonov and Agrawal, 2004] Dmitri Asonov and Rakesh Agrawal. Keyboard Acoustic Emanations. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–11, 2004.

[Atkinson, 2005a] Kevin Atkinson. *GNU Aspell*, 2005. http://aspell.sourceforge.net/.

[Atkinson, 2005b] Kevin Atkinson. *Spell Checker Oriented Word Lists*, 2005. http://wordlist.sourceforge.net/.

[Barreno *et al.*, 2006] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can Machine Learning Be Secure? In *Proceedings of the ACM Symposium on InformAtion, Computer, and Communications Security (ASIACCS'06)*, pages 16–25, 2006.

[Bernstein, 2005] Daniel J. Bernstein. Cache-timing Attacks on AES, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[Bilmes, 1997] Jeff A. Bilmes. *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. Technical Report ICSI-TR-97-021, International Computer Science Institute, Berkeley, California, 1997.

[Binkley and Singh, 2006] James R. Binkley and Suresh Singh. An Algorithm for Anomaly-based Botnet Detection. In *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 43–48, 2006.

[Briol, 1991] R. Briol. Emanation: How to Keep Your Data Confidential. In *Proceedings of Symposium on Electromagnetic Security For Information Protection*, pages 225–234, 1991.

[Broder *et al.*, 1997] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic Clustering of the Web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, 1997.

[Brodsky and Brodsky, 2007] Alex Brodsky and Dmitry Brodsky. A Distributed Content Independent Method for Spam Detection. In *HotBots'07: Proceedings of the 1st workshop on Hot Topics in Understanding Botnets*, pages 18–27, 2007.

[Brumley and Boneh, 2003] David Brumley and Dan Boneh. Remote Timing Attacks are Practical. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 1–13, 2003.

[Canvel *et al.*, 1993] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Timing-based Attacks on SSL/TLS with CBC Encryption, 1993. http://www.openssl.org/news/secadv_20030219.txt.

[Chan and Lippmann, 2006] Philip K. Chan and Richard P. Lippmann. Machine Learning for Computer Security. *Journal Machine Learning, Vol. 7*, pages 2669–2672, 2006.

[Childers *et al.*, 1977] D. G. Childers, D. P. Skinner, and R. C. Kemerait. The Cepstrum: A Guide to Processing. In *Proceedings of the IEEE, Vol. 65, No. 10*, pages 1428–1443, 1977.

[Cooke *et al.*, 2005] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 39–44, 2005.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[Dagon *et al.*, 2006] David Dagon, Cliff Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, pages 226–240, 2006.

[Daswani *et al.*, 2007] Neil Daswani, Michael Stoppelman, and the Google Click Quality and Security Teams. The Anatomy of Clickbot.A. In *HotBots'07: Proceedings of the 1st workshop on Hot Topics in Understanding Botnets*, pages 80–91, 2007.

[Dean and Ghemawat, 2004] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 137–150, 2004.

[Diffie and Hellman, 1976] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory, Vol. IT-22, No. 6*, pages 644–654, 1976.

[Fine *et al.*, 1998] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning, Vol. 32, No. 1*, pages 41–62, 1998.

[Frankowski *et al.*, 2006] Dan Frankowski, Dan Cosley, Shilad Sen, Loren Terveen, and John Riedl. You are What You Say: Privacy Risks of Public Mentions. In *SIGIR'06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 565–572, 2006.

[Freiling *et al.*, 2005] Felix C. Freiling, Thorsten Holz, and Georg Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05)*, pages 319–335, 2005.

[Ghemawat *et al.*, 2003] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP'03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[Hadoop, 2007] Hadoop. Hadoop, 2007. http://lucene.apache.org/hadoop/.

[Hulten *et al.*, 2004] Geoff Hulten, Joshua Goodman, and Robert Rounthwaite. Filtering Spam E-mail on a Global Scale. In *WWW Alt.'04: Proceedings of the 13th International World Wide Web Conference on Alternate Track*, pages 366–367, 2004.

[Isard *et al.*, 2007] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Operating Systems Review, Vol. 41, No. 3*, pages 59–72, 2007.

[Jordan, 2008] Michael I. Jordan. *An Introduction to Probabilistic Graphical Models*. 2008. In preparation.

[Jurafsky and Martin, 2000] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[Karasaridis *et al.*, 2007] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale Botnet Detection and Characterization. In *HotBots'07: Proceedings of the 1st workshop on Hot Topics in Understanding Botnets*, pages 49–56, 2007.

[Kocher *et al.*, 1999] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, 1999.

[Kocher, 1996] Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, 1996.

[Krasser *et al.*, 2005] Sven Krasser, Gregory Conti, Julian Grizzard, Jeff Gribschaw, and Henry Owen. Real-time and Forensic Network Data Analysis Using Animated and Coordinated Visualization. In *IAW'05: Proceedings of the 6th IEEE Information Assurance Workshop*, pages 42–49, 2005.

[Kuhn, 2002] Markus G. Kuhn. Optical Time-Domain Eavesdropping Risks of CRT Displays. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–18, 2002.

[Kuhn, 2003] Markus G. Kuhn. Compromising Emanations: Eavesdropping Risks of of Computer Displays. Technical Report UCAM-CL-TR-577, Computer Laboratory, University of Cambridge, 2003.

[Maloof, 2006] Marcus A. Maloof. *Machine Learning and Data Mining for Computer Security: Methods and Applications (Advanced Information and Knowledge Processing)*. Springer, 2006.

[Manber, 1994] Udi Manber. Finding Similar Files in a Large File System. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[Monrose and Rubin, 2000] Fabian Monrose and Aviel D. Rubin. Keystroke Dynamics as a Biometric for Authentication. *Future Generation Computer Systems, Vol. 16, No. 4*, pages 351–359, 2000.

[Percival, 2005] Colin Percival. Cache Missing for Fun and Profit, 2005. http://www.daemonology.net/papers/htt.pdf.

[Rabin, 1981] Michael O. Rabin. Fingerprinting by Random Polynomials, 1981. Harvard Aiken Computational Laboratory TR-15-81.

[Rabiner and Juang, 1986] L. R. Rabiner and H. Juang. An Introduction to Hidden Markov Models. In *IEEE Acoustics, Speech, and Signal Processing Magazine, Vol. 3*, pages 4–16, 1986.

[Rajab *et al.*, 2006] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC'06: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pages 41–52, 2006.

[Rajab *et al.*, 2007] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My Botnet is Bigger than Yours (Maybe, Better than Yours). In *HotBots'07: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets*, 2007.

[Ramachandran and Feamster, 2006] Anirudh Ramachandran and Nick Feamster. Understanding the Network-level Behavior of Spammers. In *SIGCOMM'06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 291–302, 2006.

[Ramachandran *et al.*, 2006] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing Botnet Membership Using DNSBL Counter-intelligence. In *SRUTI'06: Proceedings of the 2nd Conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 49–54, 2006.

[Rivest *et al.*, 1983] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *ACM Communication, Vol. 26, No. 1*, pages 96–99, 1983.

[Russell and Norvig, 2003] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, 2nd edition.* Prentice Hall, 2003.

[Saponas *et al.*, 2007] T. Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices That Tell On You: Privacy Trends in Consumer Ubiquitous Computing. In *Proceedings of the 16th USENIX Security Symposium*, pages 55–70, 2007.

[ShadowServer.Org, 2007] ShadowServer.Org. Shadow Server Foundation, 2007. http://www.shadowserver.org/.

[Shamir and Tromer, 2004] Adi Shamir and Eran Tromer. *Acoustic Cryptanalysis*, 2004. http://www.wisdom.weizmann.ac.il/~tromer/acoustic/.

[Song *et al.*, 2001] Dawn Song, David Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proceeding of the 10th USENIX Security Symposium*, pages 337–352, 2001.

[Thede and Harper, 1999] Scott M. Thede and Mary P. Harper. A Second-order Hidden Markov Model for Part-of-speech Tagging. In *Proceedings of the 37th conference on Association for Computational Linguistics*, pages 175–182, 1999.

[Wasserman, 1993] Philip D. Wasserman. *Advanced Methods in Neural Computing*. Wiley, 1993.

[Wikipedia, 2008a] Wikipedia. Wikipedia: Data Encryption Standard, 2008. http://en.wikipedia.org/wiki/Data_Encryption_Standard.

[Wikipedia, 2008b] Wikipedia. Wikipedia: E-mail Spam, 2008. http://en.wikipedia.org/wiki/E-mail_spam.

[Xie *et al.*, 2007] Yinglian Xie, Fang Yu, Kannan Achan, Eliot Gillum, Moises Goldszmidt, and Ted Wobber. How Dynamic are IP Addresses? In *SIGCOMM'07: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 301–312, 2007.

[Zhou *et al.*, 2003] Feng Zhou, Li Zhuang, Ben Y. Zhao, Ling Huang, Anthony D. Joseph, and John D. Kubiatowicz. Approximate Object Location and Spam Filtering on Peer-to-peer Systems. In *Proceeding of Middleware*, pages 1–20, 2003.

[Zhuang *et al.*, 2005] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard Acoustic Emanations Revisited. In *CCS'05: Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 373–382, 2005.

[Zhuang *et al.*, 2008] Li Zhuang, John Dunagan, Daniel R. Simon, Helen J. Wang, Ivan Osipkov, Geoff Hulten, and J. D. Tygar. Characterizing Botnets from Email Spam Records. In *LEET'08: Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.